SanityCheck™

User Manual

SanityCheck™

User Manual

©1994-2001 Committed Software. All Rights Reserved.

Published World-Wide by Committed Software.

SanityCheck

Written by: Paul Carnine

Manual: Paul Carnine & Rich Gay

Software License and Limited Warranty

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE CONTAINED ON THE DISKS. BY USING THE SOFTWARE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT, WHICH INCLUDES THE SOFTWARE LICENSE AND WARRANTY DISCLAIMER (collectively referred to herein as the "Agreement"). THIS AGREEMENT CONSTITUTES THE COMPLETE AGREEMENT BETWEEN YOU AND COMMITTED SOFTWARE, INC. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT USE THE SOFTWARE AND PROMPTLY RETURN THE PACKAGE FOR A FULL REFUND.

1. Ownership of Software. The enclosed manual and computer programs ("Software") were developed and are copyrighted by Committed Software and are licensed, not sold, to you by Committed Software for use under the following terms, and Committed Software reserves any rights not expressly granted to you. You own the disk on which any software is recorded, but Committed Software retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.

2. License. Committed Software, as Licensor, grants to you, the LICENSEE, a non-exclusive, non-transferable right to use this Software subject to the terms of the license as described below:

   a. You may make backup copies of the Software for your use provided that they bear the Committed Software copyright notice.

   b. The licensee, and only the licensee, may use this Software on an unlimited number of structure files. No additional product license is required.

3. Restrictions. You may not distribute copies of the Software to others (except the Demonstration version included as part of the Software) or electronically transfer the Software from one computer to another over a network. This software contains trade secrets and to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human perceivable form. YOU MAY NOT MODIFY, ADAPT, TRANSLATE, RENT, LEASE, LOAN, OR RESELL FOR PROFIT THE SOFTWARE OR ANY PART THEREOF.

4. Termination. The license is effective until terminated. This license will terminate immediately without notice from Committed Software if you fail to comply with any of its provisions. Upon termination, you must destroy the Software and all copies thereof, and you may terminate this license at any time by doing so.

5. Update Policy. Committed Software may create, from time to time, updated versions of the Software. At their option, Committed Software will make such updates available to the Licensee.

6. Warranty Disclaimer: THE SOFTWARE IS PROVIDED “AS IS” WITHOUT WARRANT OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. COMMITTED SOFTWARE DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIALS IN THE TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS IS DEFECTIVE YOU, AND NOT COMMITTED SOFTWARE OR IT'S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES, ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. However, Committed Software warrants to the original Licensee that the disk(s) on which the Software is recorded is free from defects in materials and workmanship under normal use and service for a period of thirty (30) days from the date of delivery as evidenced by a copy of the receipt.

7. THIS IS THE ONLY WARRANT OF ANY KIND, EITHER EXPRESS OR IMPLIED, THAT IS MADE BY COMMITTED SOFTWARE ON THIS SOFTWARE PRODUCT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY COMMITTED SOFTWARE OR IT'S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON SUCH INFORMATION OR ADVICE. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

8. Governing Law. This Agreement shall be governed by the laws of the State of California.

SanityCheck, SanityCheck logo are trademarks of Committed Software
Apple and Macintosh are registered trademarks of Apple Computer, Inc.
4th Dimension, 4D Compiler, 4D External Mover, and 4D Mover are registered trademarks of ACI and ACI US, Inc.

Thanks

Software License and Limited Warranty

All of you who have made SanityCheck a success, especially the beta testers.
David Beaver and Carrie Johnson who took me from the hell of L.A.
and offered me a great environment to work in.
Rich Gay & the folks at Foresight Technology for publishing SanityCheck
and for the hard work that went into version 2.
Lady G's African Cuisine, Oakland CA, for the best damn Corned Beef Stew
ever made.
BMW for making my motorcycle; it saved my sanity.

Copyright © 1994 -1999 Committed Software

Table of Contents

What's New in SanityCheck v4

This chapter describes many of the new features and capabilities that are new in SanityCheck v4.0.

Compatibility with 4D v2, v3, and v6, v6.7 (Components) for both Macintosh and Windows

SanityCheck is compatible with all versions of 4D since version 2, including PC native files, and handles byte-swapped PICT verification. SanityCheck now properly handles v6.7 "Components"

Analysis

SanityCheck now scans for just under 400 different types of problems / issues.

- syntax checking
- structure file damage checking
- resource file checking (4DX folders)
- proprietary PICT processing for finding damaged pictures.
- resource AND object cross reference checking.
- every major object within your 4D file.

New Component Features

v6.7 of 4D introduced the concept of Components to 4D. Here is a list of the component related features in v4:

- Components are now parsed and verified.
- Component Report added to "Special" menu to display a complete list of all components and the contents of those components.
- Component Template available in SanityCheck explorer, which allows detailed display of the Component resource (mod4).
- Component damage detection added.
- Non-component compresson detected (ie, incorrectly encrypted/ compressed objects are detected).

SanityCheck Plugins

v4 of SanityCheck has support for SanityCheck plugins. This plugin API

was created to start to answer the question of "Naming conventions" within 4D. You can write your own plugin to SanityCheck to support your own naming conventions.

- API is released publically.
- API supports "object naming" so that object names can be scanned and tagged.
- VariableNamer, a sample SanityCheck plugin, will be released with API. VariableNamer only does name checking of variables, although all object names (tables, forms, etc) are offered via the API.

Respects Mac4DX / Win4DX in "system" directories.

v4 of SanityCheck will read 4D plugins from the "system" directories.

Comment Searching

Comments are now fully cross referenced and searchable via the Explorer interface.

Method Parameter Checking

Caller <--> Callee method parameter checks are done to verify that the proper number of parameters are passed between routines.

The Compiler_ routine definition of a method is also compared against the actual method's usage of parameters.

Unused / Double Declared Variables detected

Variables that are defined, but never used are now detected. This was an oft-requested feature to help users find dead-variables.

Variables that are defined twice (double declared) are also reported by SanityCheck.

Objects that have no style sheets detected

Objects that have no style sheets defined are detected. This is very useful feature for those who are implementing cross-platform data-bases, as they can quickly find areas where style sheets are not being used (and may have visual display problems on the secondary plat-form).

Display of unreported Items

The SanityCheck log now displays which options were "disabled" during the run. This helps users to remember which items were turned off during a particular scan.

Detect 30 char limit on form variables (TechTip)

There is a 4DTechTip (http://www.4d.com/support/tips00-395.html), Tim Tonooka, August 2000, which states that there is a 30 char limit on variable names on forms.

SanityCheck detects variables with 31 characters.

Other minor features:

In addition, here are other features added to verson 4 of SanityCheck:

- Detect "Damage" reported by 4D Tools when Form Input or Output is not selected. This is not actually damage, but 4D Tools reports it as such, so SanityCheck now reports it as a warning.
- Detect missing colons on "case of" statements.
- detect missing methods for: EXECUTE ON SERVER, EXECUTE ON CLIENT, ON ERR CALL, ON EVENT CALL, NEW PROCESS, SET ABOUT.

How to Operate SanityCheck

This chapter tells you the basics you'll need to know about how to operate SanityCheck. There are also shortcuts that will help you work faster with SanityCheck.

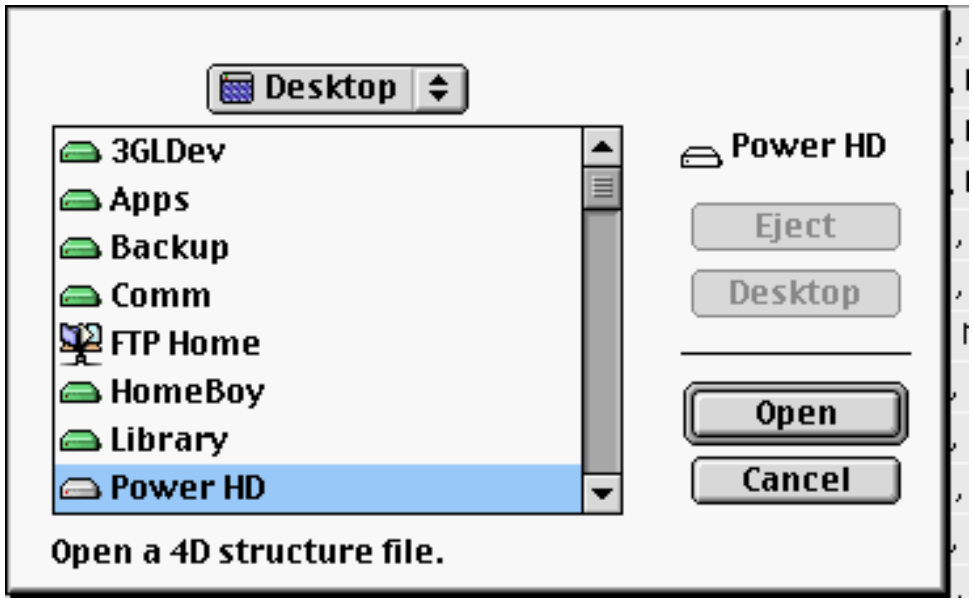
Installing SanityCheck

SanityCheck needs no special installer — Simply drag the Sanity-Check icon from the SanityCheck disk onto your Macintosh hard disk. You may run SanityCheck from the SanityCheck disk, but SanityCheck will run much faster when it is run from a hard disk.

Launching SanityCheck

To Launch SanityCheck by Double-clicking

1 Double click on the SanityCheck icon to launch SanityCheck. The Open File dialog is displayed.



SanityCheck is assuming that you wish to open a structure file and is therefore prompting you to locate one on your disk.

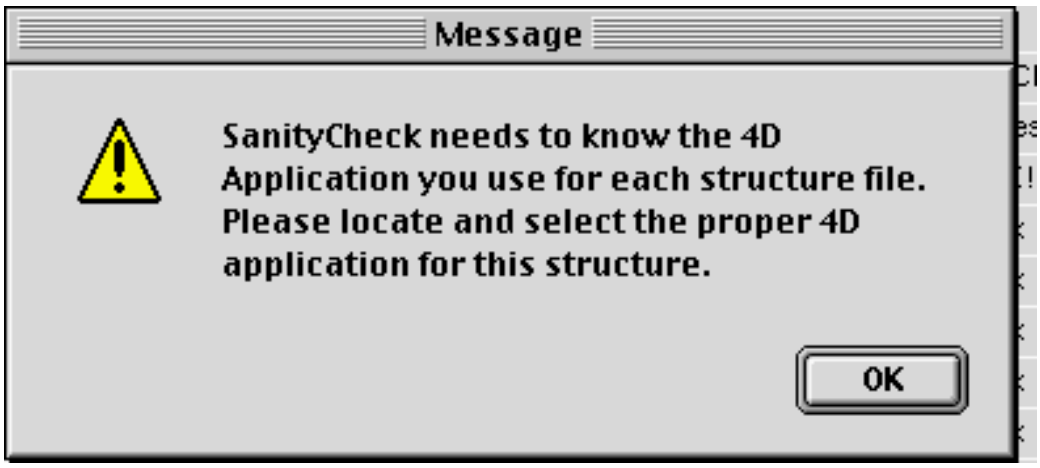
2 Select the structure file you wish to open.

3 Click the Open button

To Launch SanityCheck via Drag-and-Drop

1 In the Finder, drag-and-drop your structure file onto Sanity-

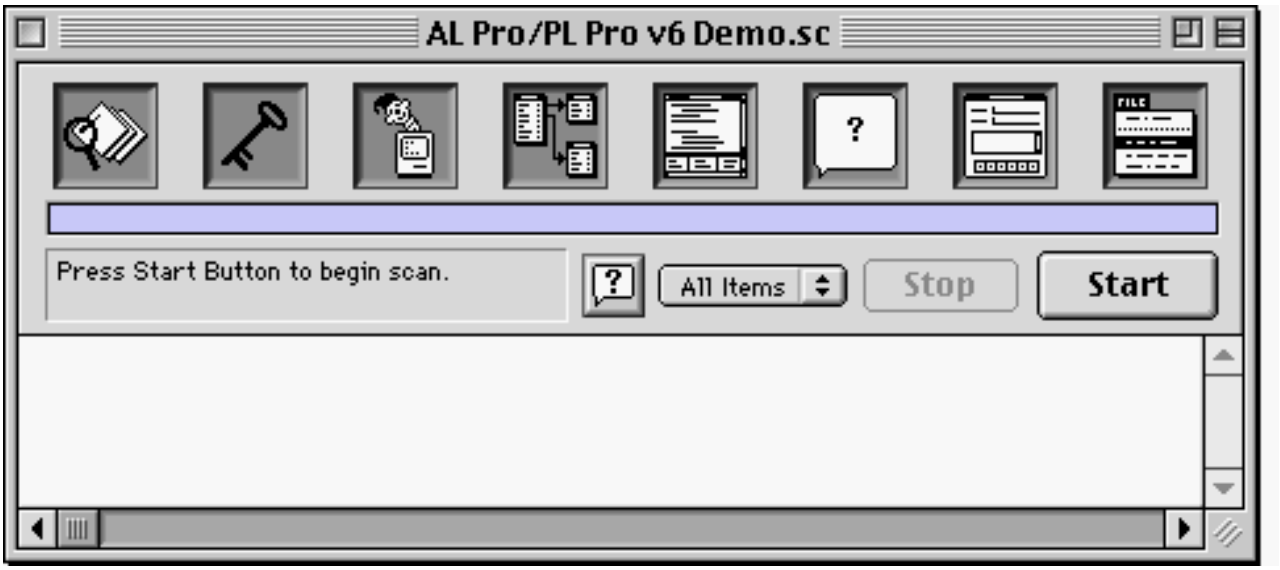
Check. SanityCheck will accept a drag-and-drop of a structure file at any time, as long as there is enough memory to open your the file.
SanityCheck will prompt you to select the 4D application used with this structure file.



2 Select the 4D application last used to open the structure file you've previously selected.

Running SanityCheck

Once your structure file is opened by SanityCheck, a window is opened which lets you select general scan options, and initiate a scan. The window also displays the results of the scan, called the "log" of the scan, in the bottom area of the window.



Note: this window combines the scan and log windows previously displayed by SanityCheck v1 and v2.

To Scan a Database File

1 Click the Start button.
SanityCheck will immediately start performing an integrity check and scan for programming errors on your structure file. If you would like to utilize the Find features, please refer to the section “Using the Find Dialog” on page 17.

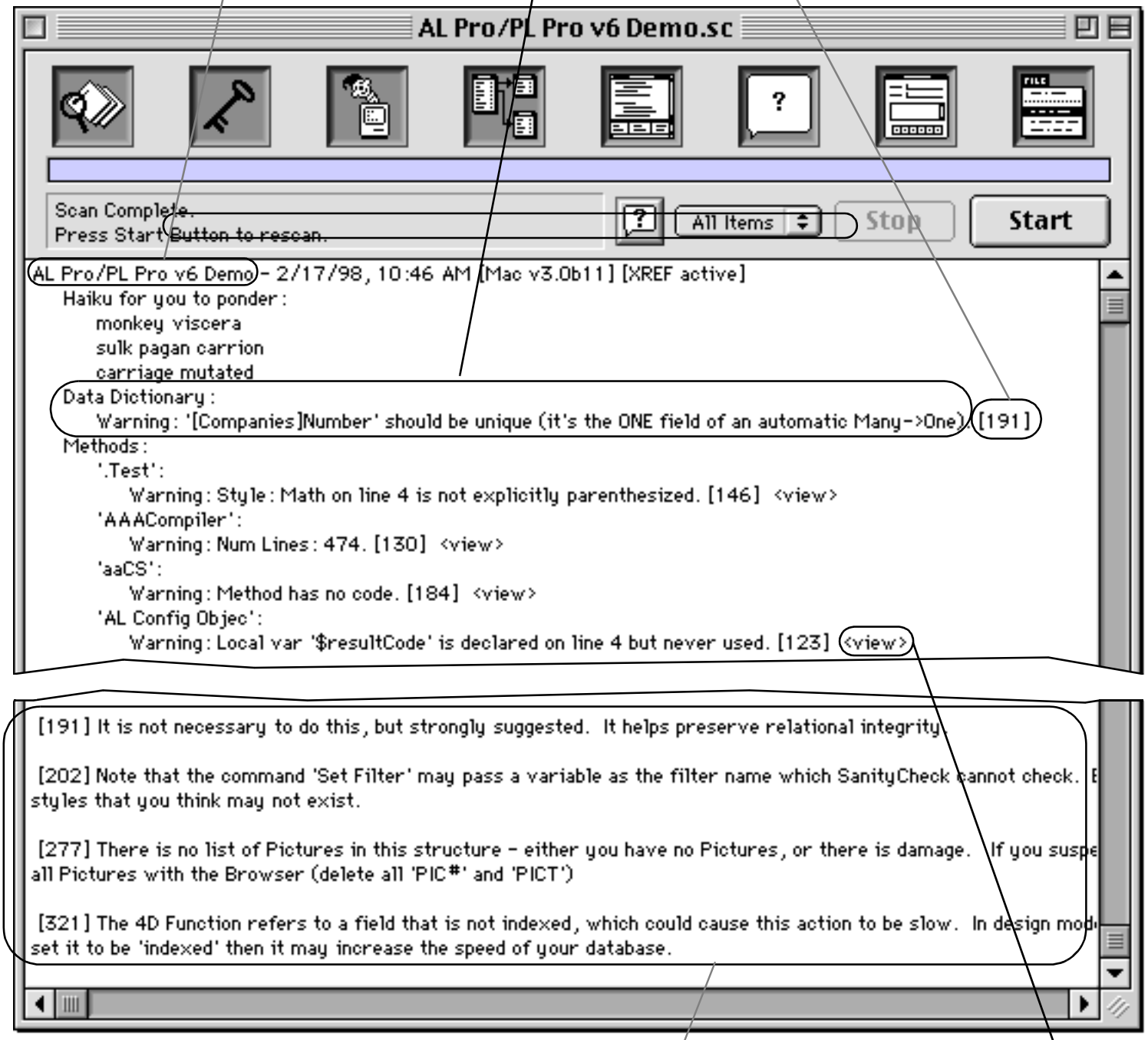
To Close a Structure File

1 Click the Close Box or select the Close command in the File menu when the structure file is the topmost window.
See “Common Questions” on page 44 for details on how Sanity-Check manages files.

Using the Log File

As SanityCheck scans a structure file and encounters an item that needs to be reported, the log area of the window will display the items as the scan progresses.

The log file has two parts, the item listings and the item descriptions (see below). The upper portion of the log file contains item by item of all the items that were found in your structure file.



The key to reading the item listing is to remember that the indentation describes a hierarchy, very much like a genealogy tree.

In the example above, the seventh line in the log (begins with ‘Warn-

ing: '[Companies]Number...'), should be read as: *'*Companies]Number*', within 'Data Dictionary' (the structure file definition of the database tables and fields), should be a unique field.*

Note: Although the statistics items that appear in the log do have item numbers (so you can turn them off), they do not appear in the bottom half of the log file.

To Print the Log File

1. Make sure the foremost window is a log file.
2. Choose the Print command in the File menu.

HINT: If you click the “Zoom” box (located in the upper right corner of the window) SanityCheck will set the window to be the best size to print for the current printer and Page Setup options.

Using the Log File’s Hypertext Buttons

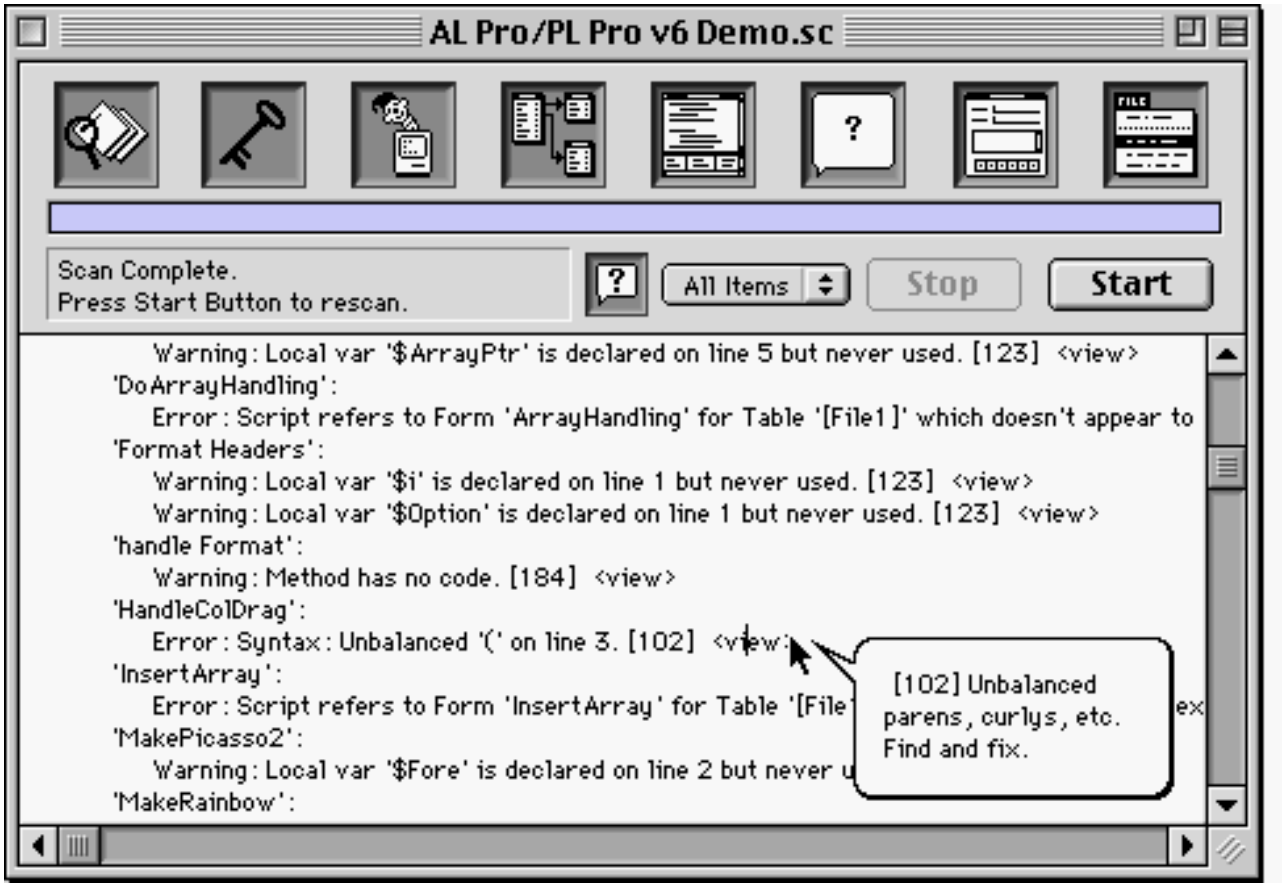
The log file contains hypertext buttons next to many reported items. You can click a hypertext button to display the object referenced in the reported item.

Using Balloon Help for Reported Items

You can enable the display of balloon help for reported items. The balloon help will contain the same detailed information for a particular item as shown at the bottom of the SanityCheck log. Using the balloon help feature lets you see the detailed explanation while viewing the reported item, without having to scroll to the bottom of the log.

To use Balloon Help for Reported Items

1. Click the Help button (the question mark).
   This button toggles the balloon help feature.
2. Position the cursor over any reported item.
   SanityCheck will display the detailed information for that item.

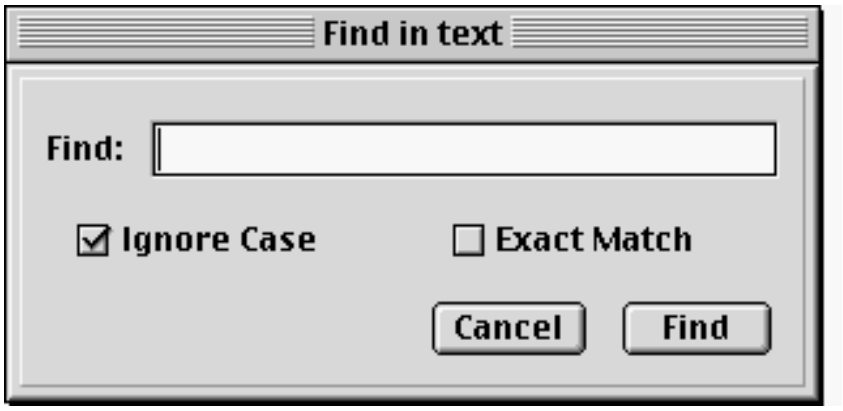


Using the Find Dialog

You can use SanityCheck's Find dialog to find occurrences of the text you specify within a log window.

To Use the Find Dialog

1. Make sure that a structure file is open and the scan/log window is the topmost window.
2. Select the Find command in the File menu, or press Command-F.
   The Find dialog is displayed.



3 Type the text you want to Find.
4 Click the Find button.
SanityCheck will find the first instance of the text you specify. To find additional instances, use the Find Again menu item.

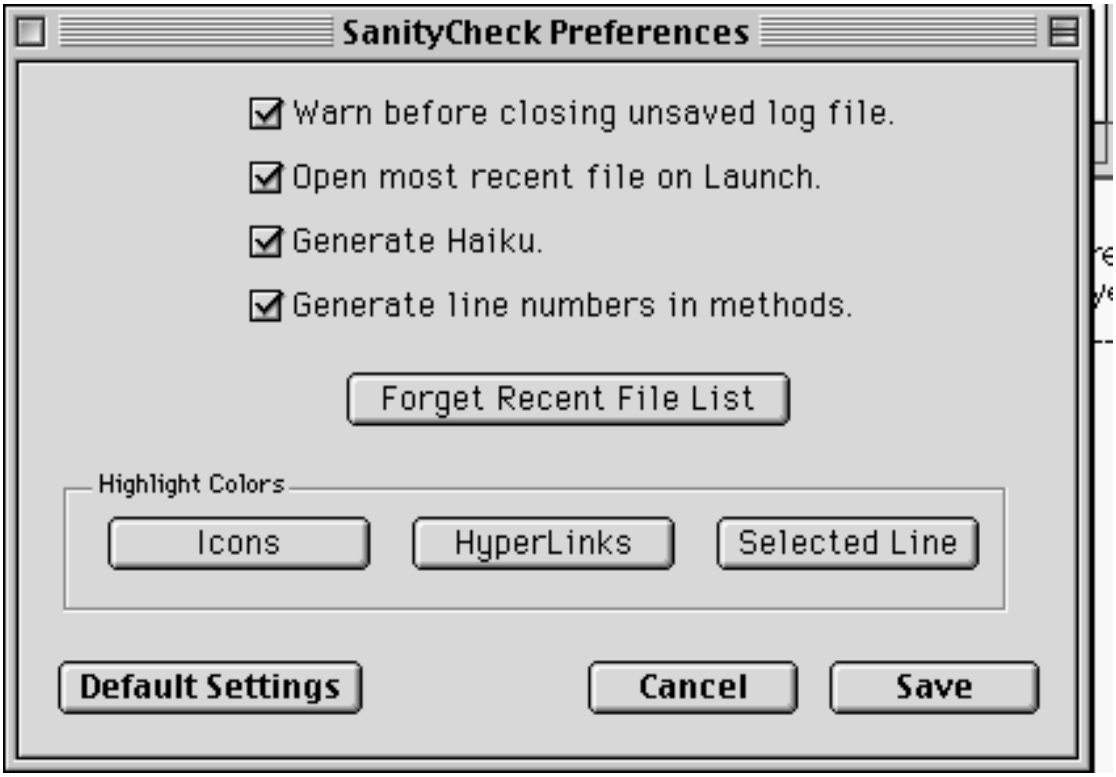
You can find objects by type, literal string, and moe using the Sanity-Check Explorer. *Please read the section* *"How to Use the SanityCheck Explorer" on page 37 for more information.*

Specifying SanityCheck Preferences

The Preferences dialog contains items that control the overall behavior of SanityCheck.

To Modify SanityCheck Preferences

To modify the Preferences, choose the Preferences command in the File menu. The Preferences dialog is displayed.



To Modify a Preference

1. Choose Preferences in the File menu.
   The Preferences dialog is displayed.
2. Click the checkbox of the preference you want to change.
3. Click the OK button.

Specifying SanityCheck Settings

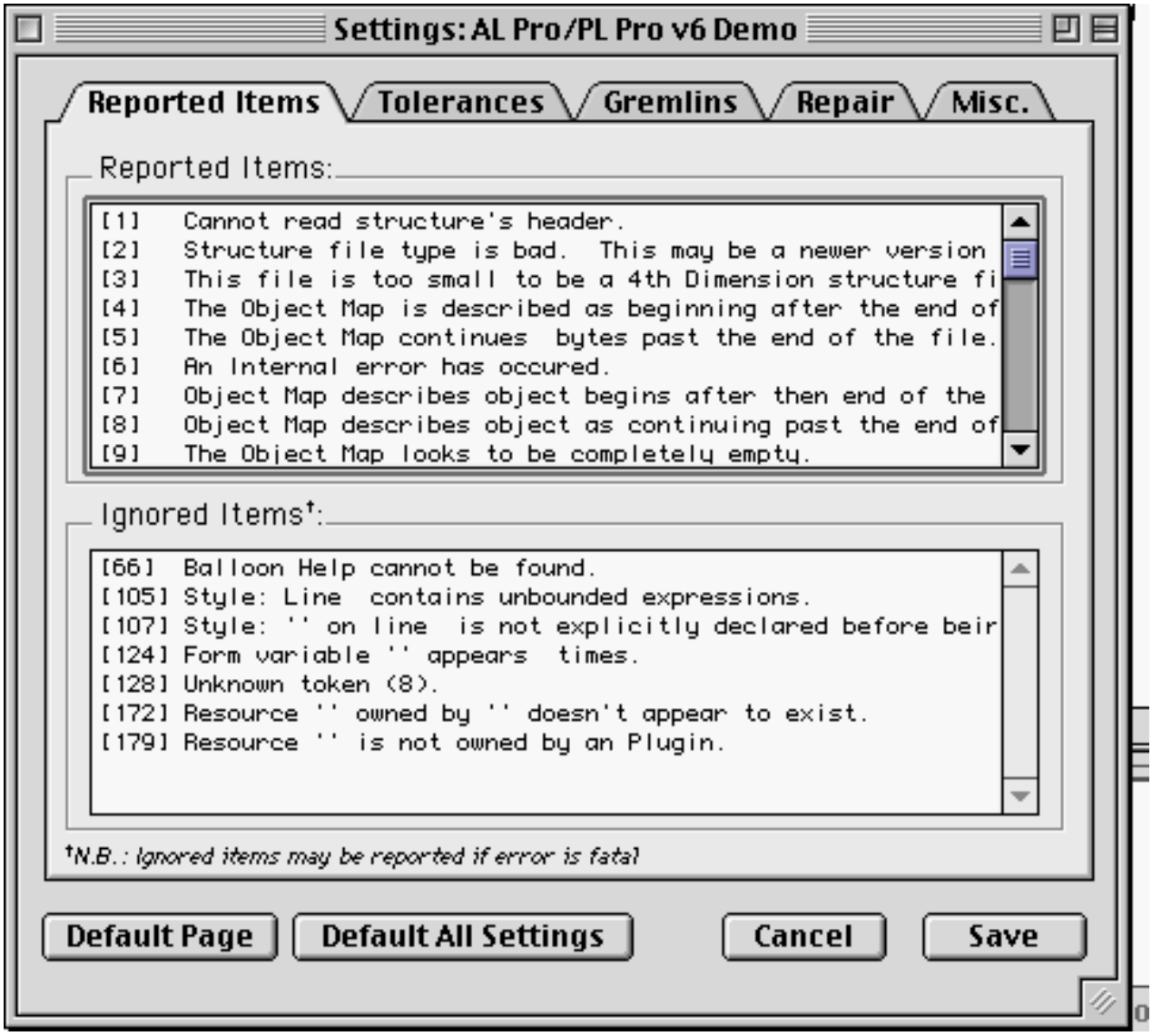
You may modify much of the analysis and reporting of Sanity-Check's operation using the Preferences and Settings dialogs.

To Modify SanityCheck Settings

1. Choose the Settings command in the File menu, or type command-semicolon. The Settings dialog is displayed.
2. Select the tab for the setting topic you are interested in.

Reported Items

SanityCheck can identify and report hundreds of items as the result of a scan. You can configure SanityCheck to ignore any of these items, using the Reported Items panel in the Settings dialog.



To Move a Reported Item from One List to the Other

1. Select the item you wish to move.
   You can select an item with the mouse, or by typing the number of the item, or by using the arrow keys. You can switch which list is active via the tab key.
2. Drag the item to the other list, or press the return key. You can also just double-click on any item and it will go to the other list.

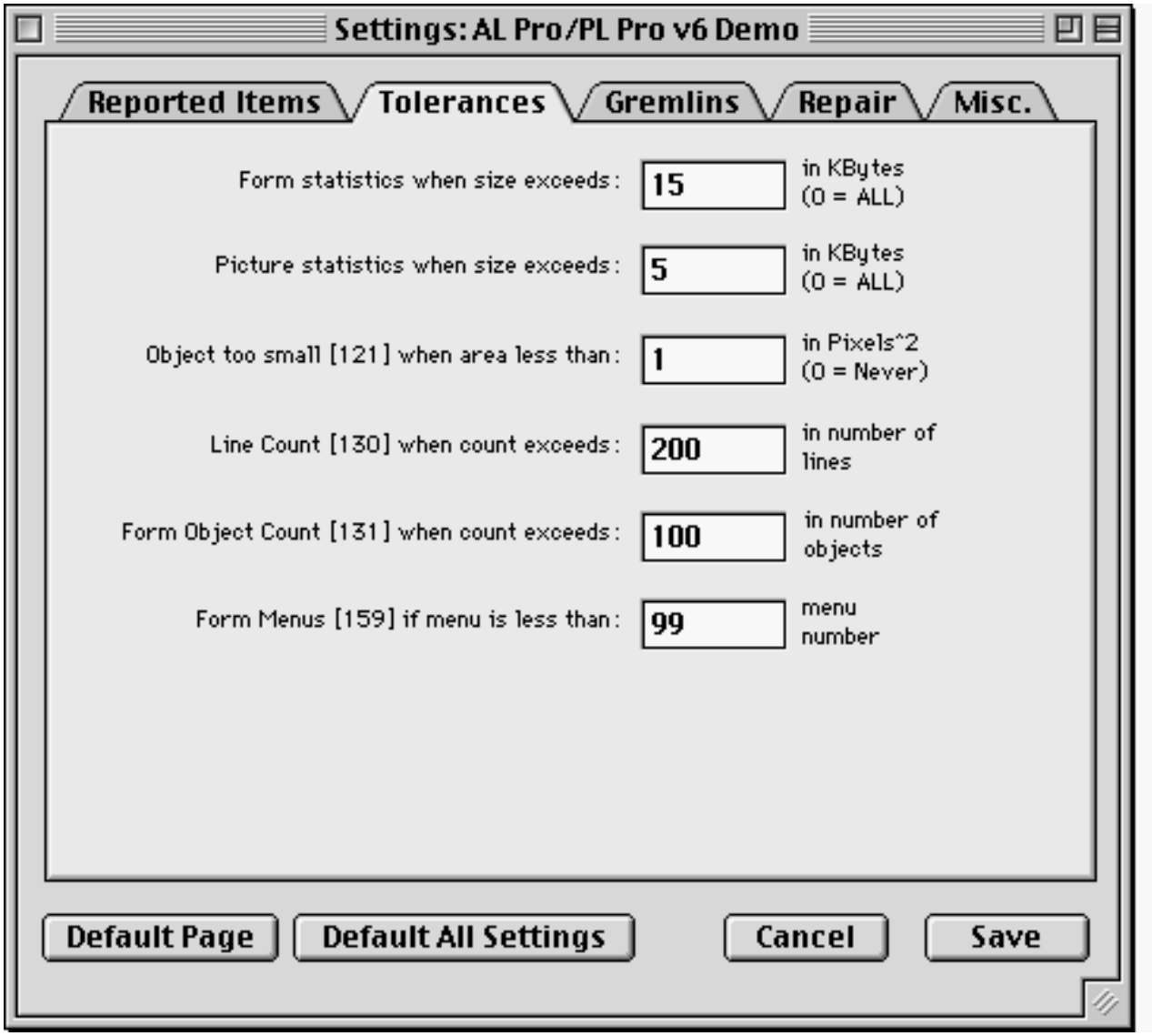
The Default Settings button will restore this screen's settings to the way they were when you first installed SanityCheck.

Tolerances

Some reported items rely on certain tolerances.

To Modify the Tolerances

1 Choose the Tolerances command in the File menu, or press command-T.
The Tolerances dialog is displayed.



2 Enter a number in the appropriate area.

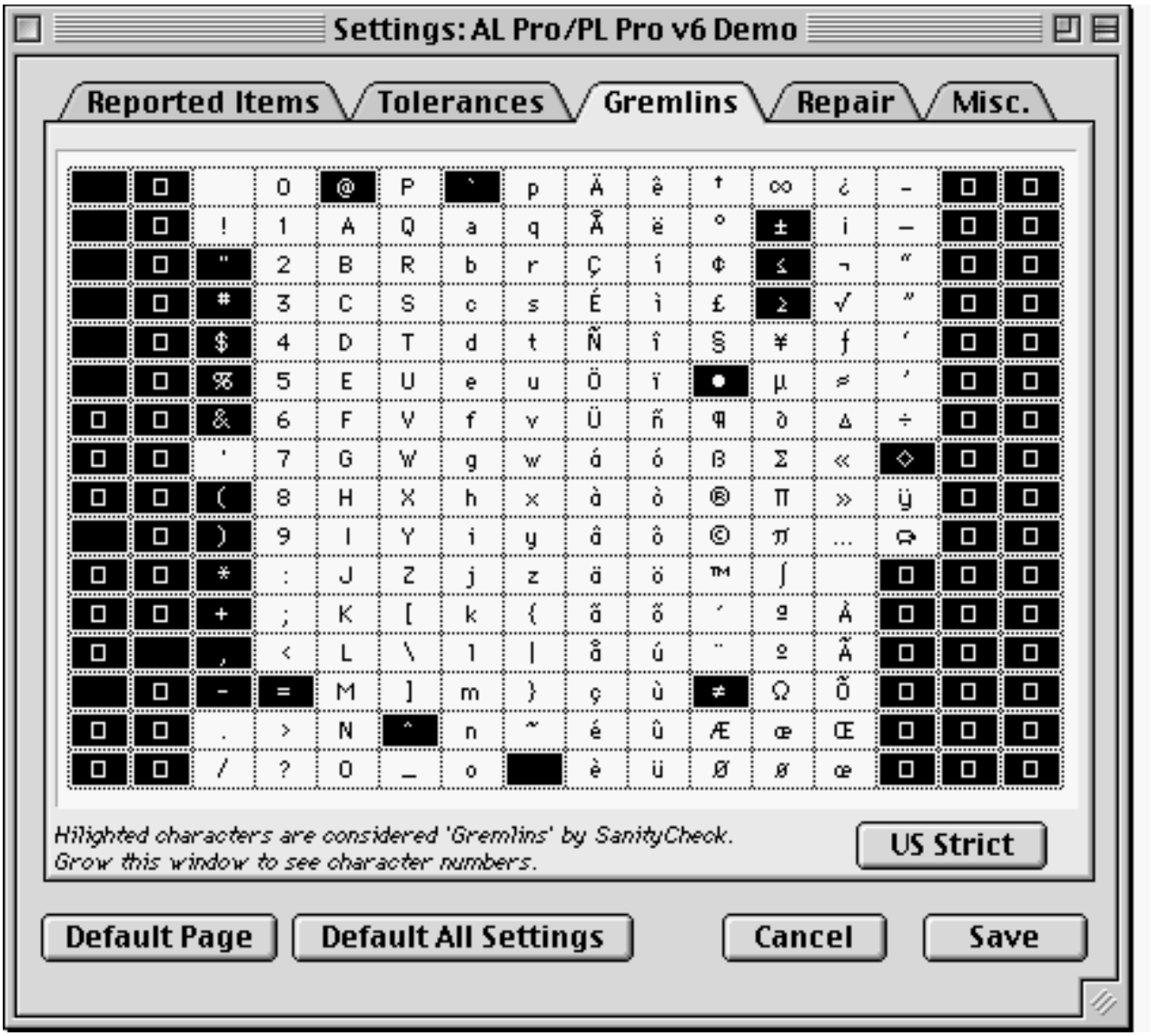
Be aware that you must make the items show up in the Reported Items panel for these tolerances to have any affect during a scan. To help you find the items in the Reported Items dialog, the item numbers affected by the tolerances are displayed in this dialog.

Gremlins

4th Dimension can sometimes have difficulty with certain extended-

ASCII characters. You can use SanityCheck to identify these characters, termed *gremlins*, which may exist in your methods or other objects, to enable you to remove them.

You can set which ASCII characters to identify as gremlins using the Gremlins tab on the Settings dialog.

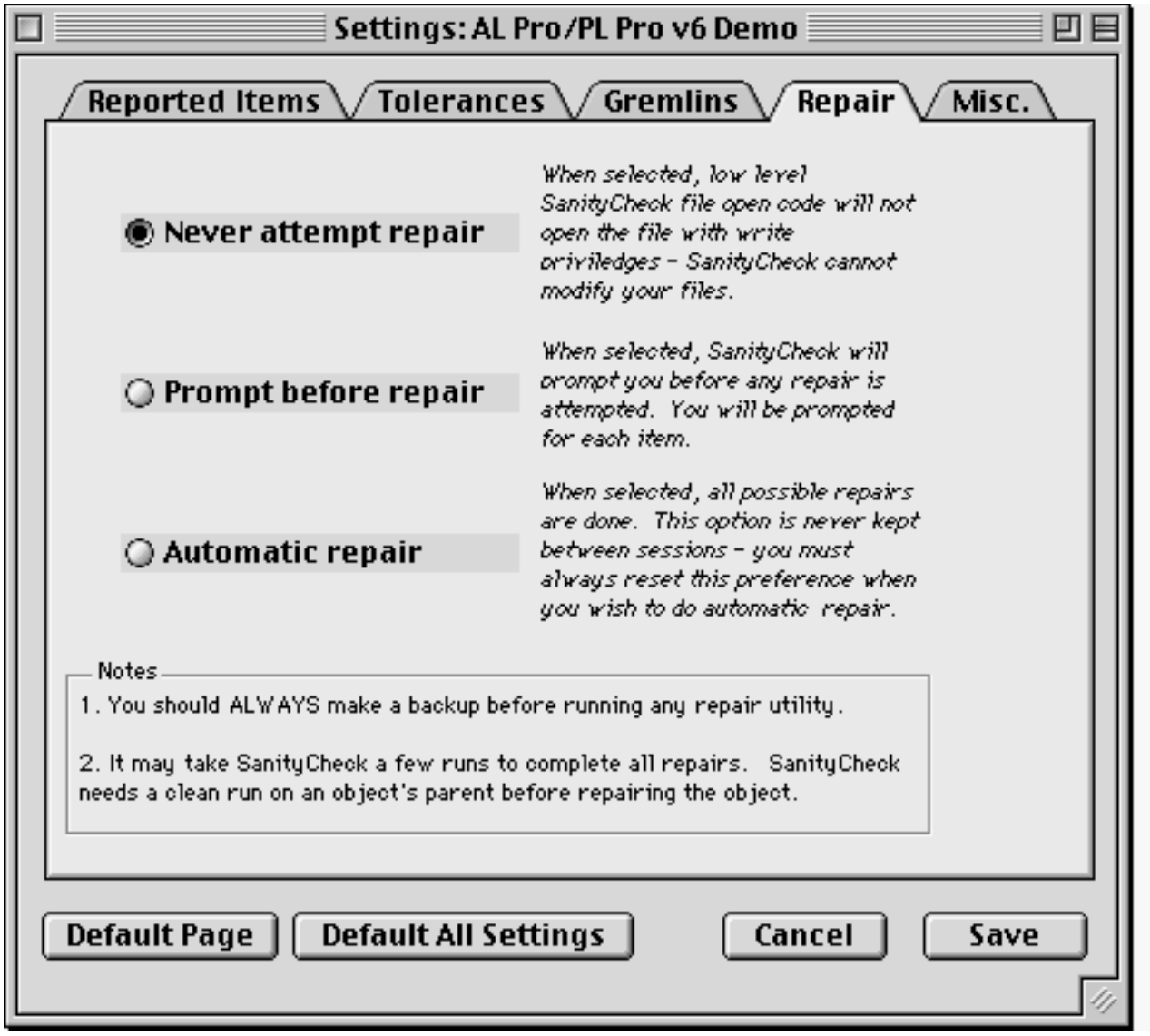


Repair

SanityCheck has the capability to repair certain kinds of problems which it identifies during a scan of a 4D structure file. You can choose to have SanityCheck optionally perform the repair operation in an automatic or semi-automatic mode.

Note: always create a backup of a structure file before attempting to

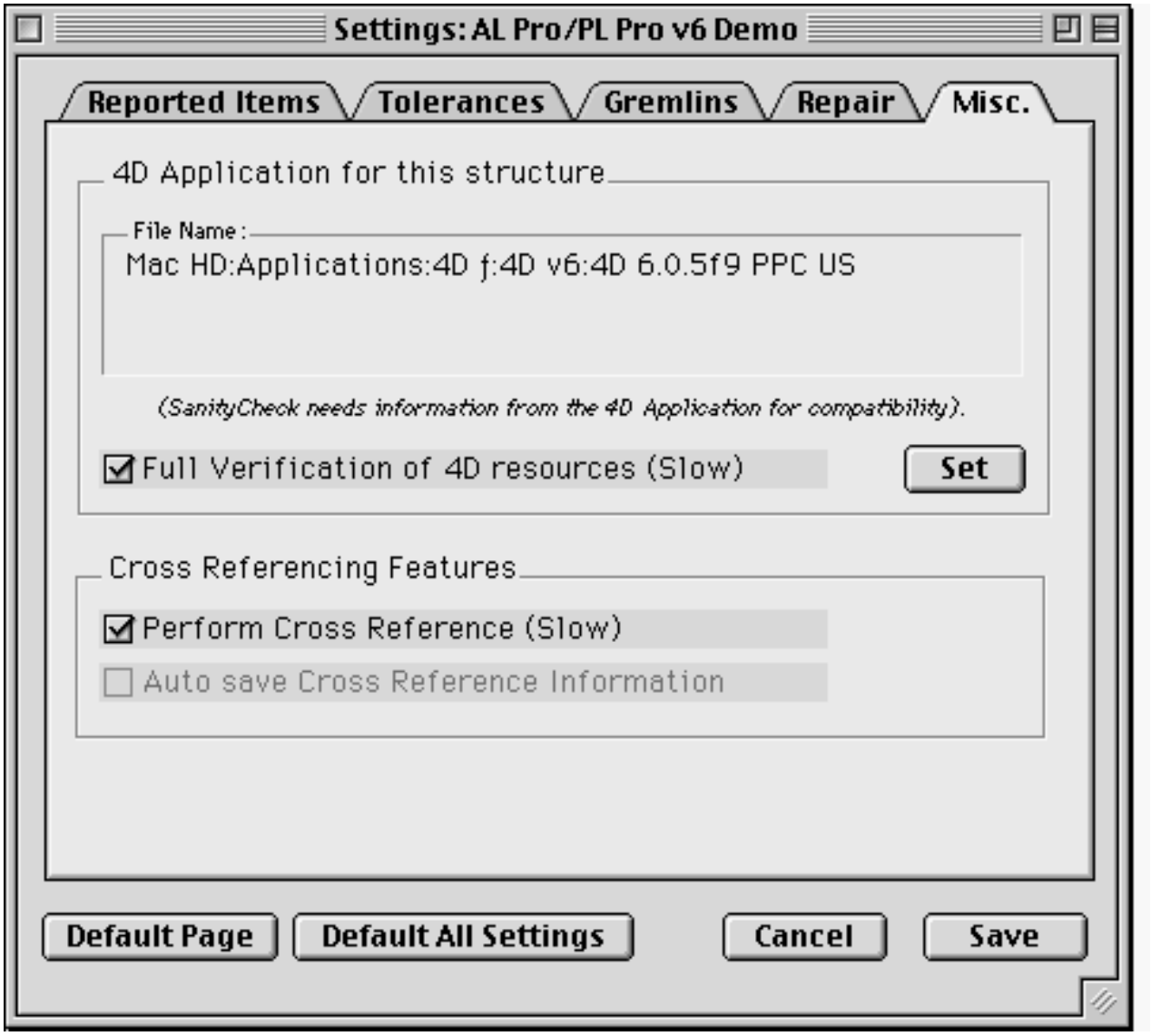
make any modification or repair to the file using SanityCheck.



Miscellaneous Settings

A few additional settings are available via the Misc. tab in the Settings dialog.

The 4D application used to open the structure file needs to be identified, and this is where you do this.



To Identify the 4D application used for the current structure file

1 Select the Settings command from the File menu.
2 Click the Misc. tab.
3 Click the Set button.
SanityCheck will prompt you to identify the 4D application used with this structure file.
4 Select the 4D application and click OK.

You can also choose to perform full verification of 4D resources. While this setting will result in longer scan times, it provides a more complete analysis which may be beneficial when trouble-shooting problems.

The cross-referencing features are used to enable operation of the

Explorer dialog. *Please read the section* *"How to Use the Sanity-Check Explorer" on page 37 for more information.*

Using SanityCheck's Common Settings.

Starting with version 4 of SanityCheck is the capability of storing a set of common settings which can be share amongst several projects.

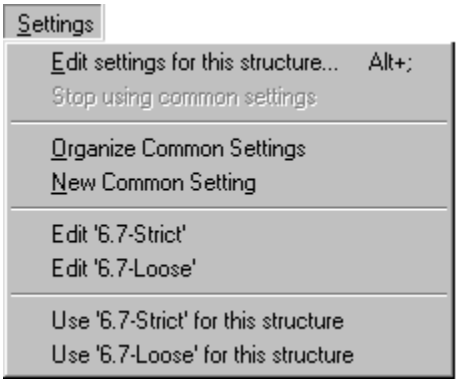
In general, you create and edit a common settings file, then for each structure file, you can specify that is uses a common settings file.

All settings are self contained in a single settings file. This includes all Plug-in settings, all tolerances, gremlin tables, etc. Therefore not only is it possible to duplicate, copy, rename, and manipulate these files, it is the preferred way of working with common settings files.

If you do not wish to use common settings, you do not have to. SanityCheck still maintains a seperate settings file for each structure file.

It is strongly recommended that you create common settings files for different versions of 4D. Among other reasons, SanityCheck stores the information about which specific executable of 4D matches this structure file in the common settings file. Since many other options may also be tied directly to a specific version of 4D, you should name your common settings like "v6.7 Strict" or "v6.7 Loose" and so on, so that it is always clear what versions you are using.

The common settings functionality is implemented in the user interface via the Settings menu:



Edit settings for this structure...

This allows you to edit the settings file that is for this specific structure file. Since SanityCheck keeps a settings file per structure, this option allows you to edit the settings in that file.

However, as soon as you start using a common settings file for the current structure, this option becomes grayed out, and the “Stop using common settings” option becomes active.

Stop using common settings.

This allows you to stop using a common settings for the current structure. This option is only active when you are using a common settings file for the current structure.

When active, this menu item will change it’s label to: “Stop using ‘Loose’ for this structure file.” where ‘Loose’ is the name of the common settings file in use by the current structure file.

Organize common settings.

This option allows you to copy, rename, delete and otherwise organize your common settings.

This option will open the common settings directory in the Finder / Explorer, and using this you can modify the settings files as you desire. SanityCheck automatically recognizes you have modified the directory and rebuilds the menu for you.

Edit ‘<settings>’.

If you have a common settings file, then they will be listed with an “Edit” prefix, which allows you to edit the contents of the settings file.

Use ‘<settings>’ for this structure.

If you have a common settings file, then this is how to specify to use that common settings for the current structure file. When you select this, it becomes the checked item, and the name <settings> will appear in the “Stop using ‘<settings>’ for this file” menu item.

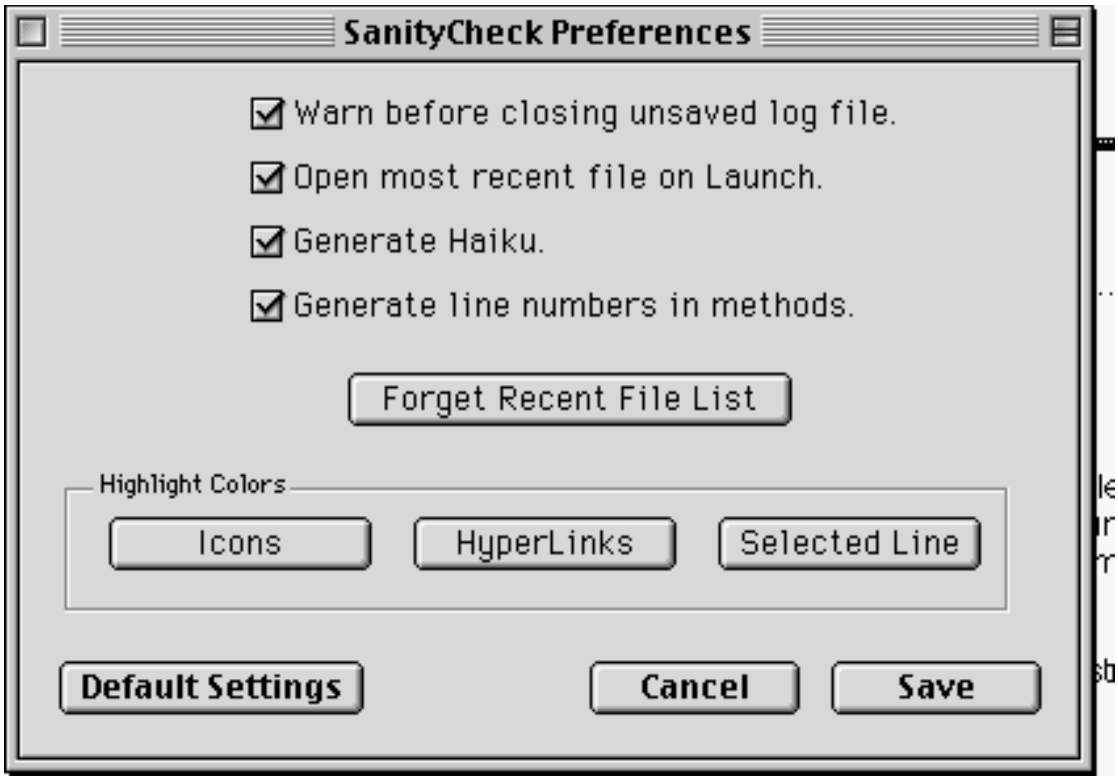
Recently Used Structures

SanityCheck automatically keeps track of the last 5 structure files that have been opened with SanityCheck. This list is available under the “File” menu. SanityCheck always sorts this list, keeping the most recently used structure at the top of the list.

You can switch to any recently used structure by selecting that structure from the File menu.

To Automatically Open the Most Recent Structure File on Launch.

1 Choosing the Preferences command in the File menu
2 Click the “Open Most recent file on launch” checkbox.



3 Click the Save button.

Clearing the Password System

You can now completely erase your 4D passwords and groups. You should be aware that this is *only* for serious damage that you may have encountered in your password area, and should be used with caution. Here are the details as to why you should rarely use this feature.

Passwords and groups are used throughout your structure file. For example, you can set up a table so that one group only has access to the table. Therefore if you delete all of your groups, 4D may still have references to the now non-existant group. This could cause serious problems to the stability of your structure file.

It is therefore suggested that you use this feature only as a last ditch effort to save your structure file that has been seriously damaged.

This feature will do the following, and nothing more: Set the number of passwords to zero, set the number of groups to zero (for both Designer and Administration passwords and groups), then make the size of the password/groups objects to refelct the smaller size. You can accomlish the same (except for resource size changes), but opening the password template and setting both password and group counts to zero.

To Clear the Passwords from a Structure File

1. Open and scan a 4D structure file.
2. Select the Clear All Passwords command from the Special menu.

Note: You must have the designer password to access this feature. It is only accessible AFTER you have given the correct designer password. If the designer password is damaged, you are out of luck and must recover from backup.

Windows Menu

The following commands are available on the Windows Menu.

Menu Command	Description
Clean Up	Neatly layers all open windows .
Close All	Closes all windows. You can also option-click in any window's close box.
list of windows	As windows are opened, their names are appended to the Windows menu. To choose any window, simply select it from the list. The windows are always sorted in order, with the frontmost window appearing at the top of the list.

How to Use the SanityCheck Browser

This chapter describes the SanityCheck Browser.

Note: the Browser is a very powerful tool, as it lets you modify nearly anything in a 4D structure file. As a result, you can easily corrupt a file if you aren't careful about your actions. Please always make backups prior to using the Browser, and be very careful about any changes you make.

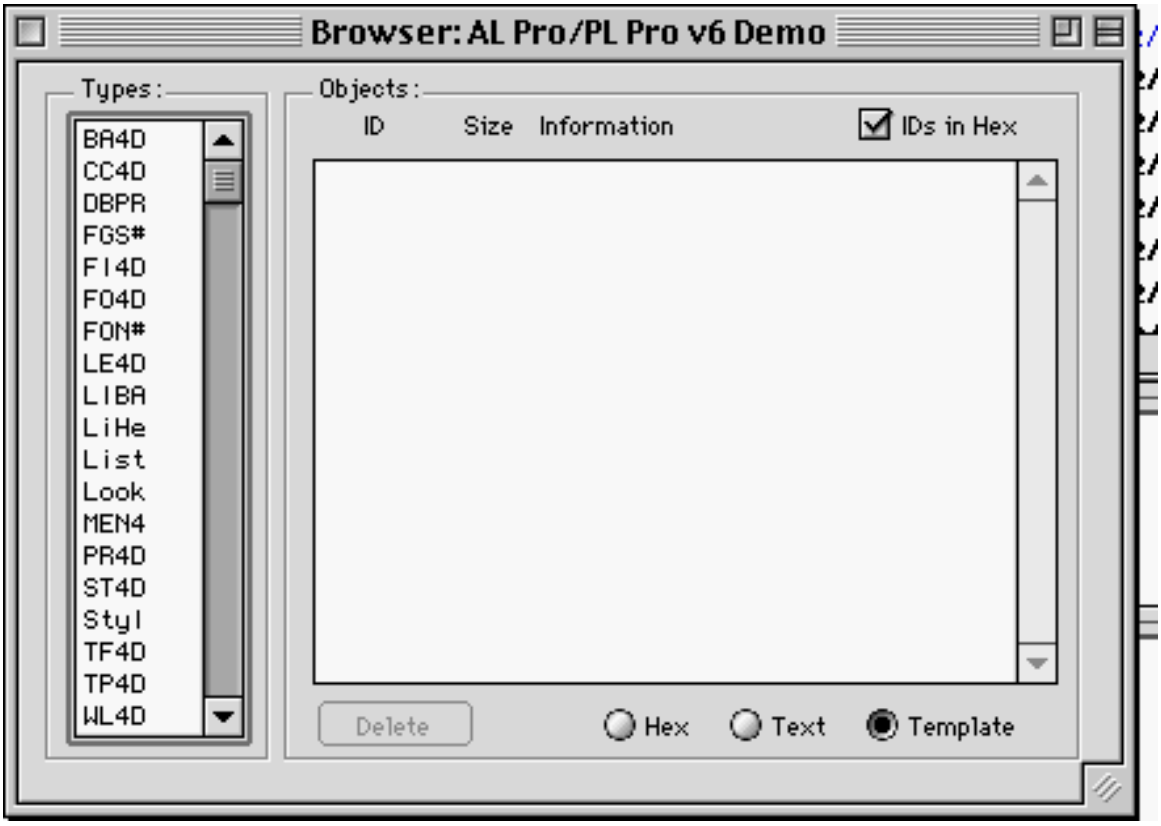
Understanding the Browser

You can think of the Browser as a ResEdit for the native 4D objects in a 4D structure file.The browser allows you to look at the raw data of your structure. We've provided templates for most of the 4D low-level object types, which you can use to edit your database structures. You should be careful in here, and make backups if you decide to make modifications.

Why is the browser available? Well, if you really do have a bad choice list, SanityCheck could fix it for you, or if you know these templates a little bit, you can fix it yourself. Just open it up and set the Choice List to 0x0000, and it's fixed. With templates, the power shifts back into your hands, out of ACI's and out of the control of the author of SanityCheck. When something goes wrong, it's much less of a black box to you. You can investigate and perhaps fix problems yourself.

To Display the Browser

1. Open a structure file.
2. Select the Browser command from the File menu.
   SanityCheck will scan the structure file (if not already scanned), and then open the Browser window.



Note that in addition to the IDs in Hex checkbox, and the Hex, Text, and Template radio buttons, you can also sort the list on the right by clicking on the ID, Size, or Information headers.

Known 4D Resource Types

SanityCheck is aware of many types of common 4D resource types, as shown in the list below.

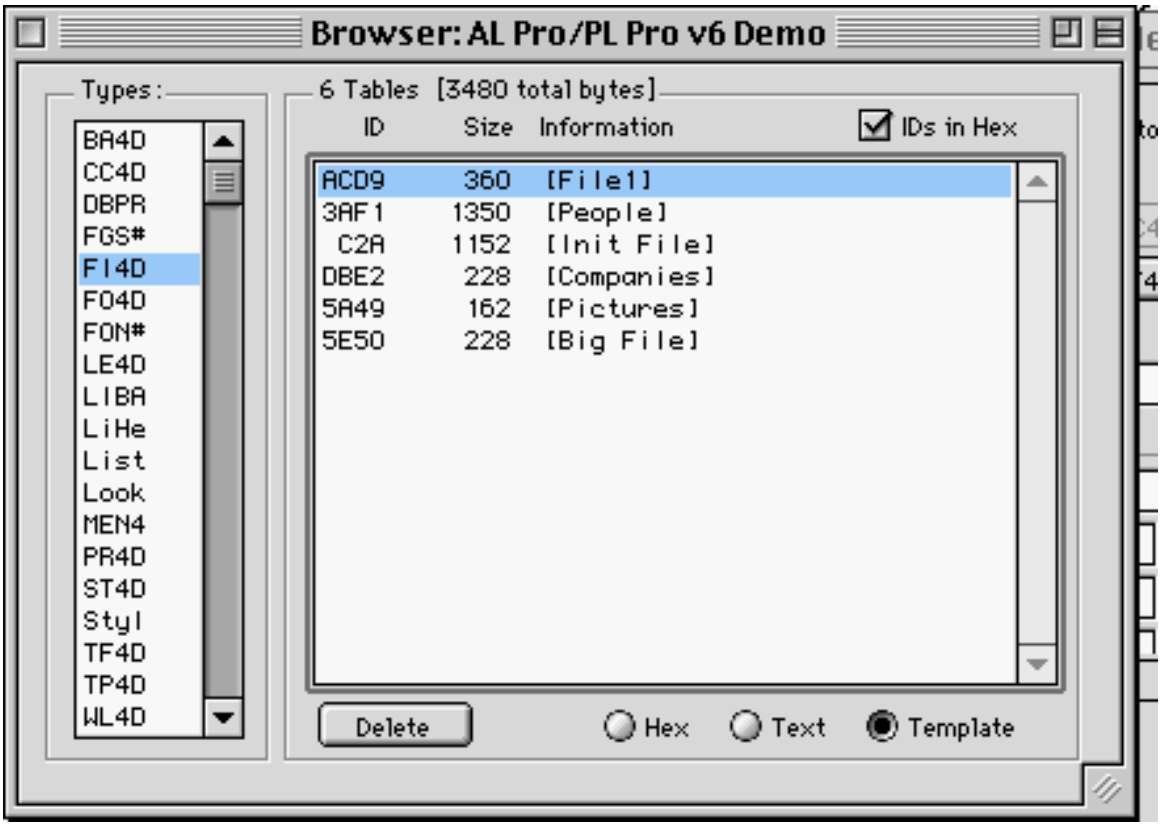
Type	Description
CC4D	Method
FO4D	Form
FI4D	Table
TF4D	Form Names
BA4D	Menu Bar
DBPR	Database Methods (OnServerStartup, etc).
FGS#	Form Template List
FON#	Font List

Type	Description
HE4D	Ballon Help Item
LE4D	List Names (can be EN4D or List)
LIBA	Menu bar List
LiHe	Balloon Help List
List	A User List (new type of EN4D)
Look	Environment preference
MEN4	A Menu
PIC#	List of pictures
PICT	A Picture
PR4D	4D Preferences
ST4D	Filter (Style)
Styl	Style definition
TP4D	Method List
WL4D	Desktop preference
WN4D	Table names
EN4D	A List (old style. see also 'List')
expl	explorer preference
FGps	Form Template
l___	window position
m___	menu window position
mem_	memory preference
pass	password window position
stru	structure window position (design environ)
xfon	platform preference

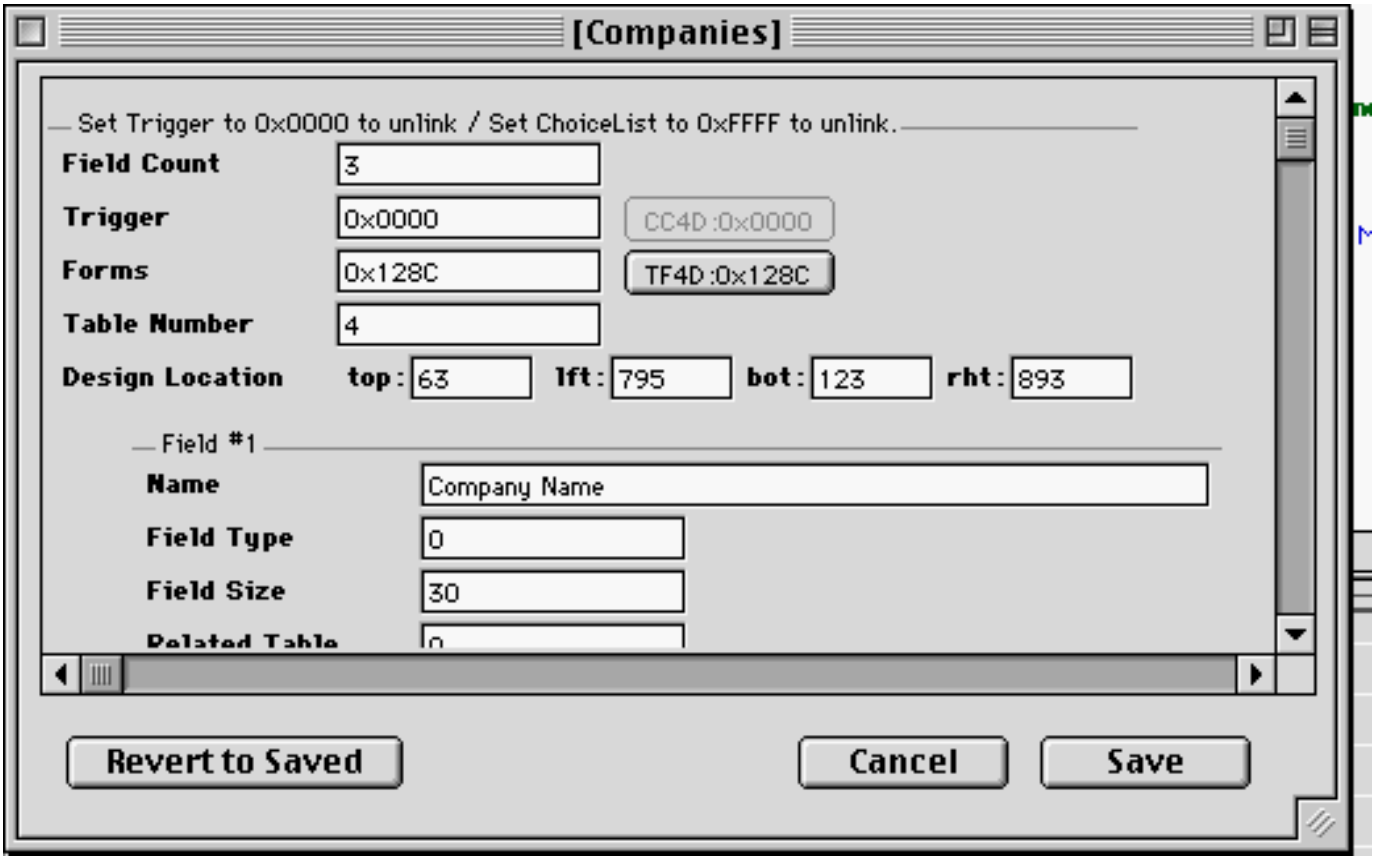
Example

One of the errors that SanityCheck identifies during a database scan is the use of a reserved word as the name of a table or field. Using the Browser, you can quickly and easily modify the name.

To do this, you would select the FI4D type, and the select the particular table the error is located in.



Double-click on the table name in the list, and SanityCheck will display a window with a form that lets you edit many of the attributes of the table, include the table name, and names of the fields in the table.



Find the field you need to modify, change the name, and then click the Save button.

Understanding Browser Templates

You can make your own templates for SanityCheck (although we will be releasing new templates as fast as we can make them). You should already know how TMPL resources work. Please refer to the documentation for ResEdit or Resourcer for more information.

The tags that are currently supported are shown in the table below. T.ags are an internal marker used by 4D to denote the start of a data structure. 4D Tools uses tags when you select the “Rebuild by Tags” options.

We plan to add additional templates over time:

Type	Description
OCNT	the counter for a list

Type	Description
LSTC	the begining of a counted list (OCNT must precede this)
LSTB	an infinately long list (ok, as long as the resource)
LSTE	the end of a list
DVDR	a divider line (mostly for a comment)
DLNG	a decimal long
DWRD	a decimal word
DBYT	a decimal byte
HLNG	a hex long
HWRD	a hex word
HBYT	a hex byte
ULNG	a hex long (note, same as DLNG for now)
UWRD	a hex word (note, same as DWRD for now)
UBYT	a hex byte (note, same as DBYT for now)
RECT	a rectangl
FWRD	a filler word
PSTR	a pascal string
Pnnn	a fixed length pascal string (note: nnn is in hex, so P010 is 16 bytes max pascal string)
Hnnn	a fixed length hex dump (note: nnn is in hex, so H020 is 32 bytes hex)
CHAR	a character

All of the following types are for SanityCheck. They are the same as HWRD above, except they denote a link.This will activate the buttons you see on the templates:

Type	Description
CC4D	a method
MEN4	a menu
TF4D	list of forms for a table
BA4D	menu bar
LE4D	list of lists
FI4D	table (with fields)

Type	Description
FO4D	a form
List	a list (could be EN4D or List)
HE4D	a help item.

Using this information and the templates in SanityCheck, you can create your own templates, if necessary.

If you create a template that you wish to share with others, you can send it to us and we will put it in a future release.

Note that we will be adding many templates to the next release of SanityCheck. Our goal is to have a template for every type.

Using the Password Template

You can modify the raw data of your passwords from within Sanity-Check. This is very useful in the case where something is damaged and is causing 4D to not behave properly.

We strongly suggest that you never change the groups or password counts. Do *not* try to delete the last password by reducing the number of passwords. The data is stored in a format such that the groups come directy *after* the passwords, and reducing the password cause the template to show invalid information for the ensuing groups.

We plan to be upgrading this area to provide a more user friendly access to the password area, however, for now, please use this area to modify information only, and not to try to remove passwords or groups.

Note: The passwords are “write-only”, meaning you can't see them or copy them. You can only change them, if you desire to.

Note: Since the browser is only active after you have entered the designer password, you must have the designer password to access the password templates.

To Display the Password Template for a Structure File

1. Open and scan a structure file.
2. Open the Browser.
3. Select the PA4D item in the Types list.
   The list on the right will display two items. The first item contains

the usernames, and the second contains the groups. Double-click on either item to edit. Follow the instructions on the form which is displayed.

Note: always maintain a backup prior to making any change to a structure file using SanityCheck, and use special care following any changes to the password setup.

Using the Browser to Delete Objects

You should rarely have to use the Browser to delete an object from your 4D file. However, if it becomes necessary, here are a few rules for you to follow:

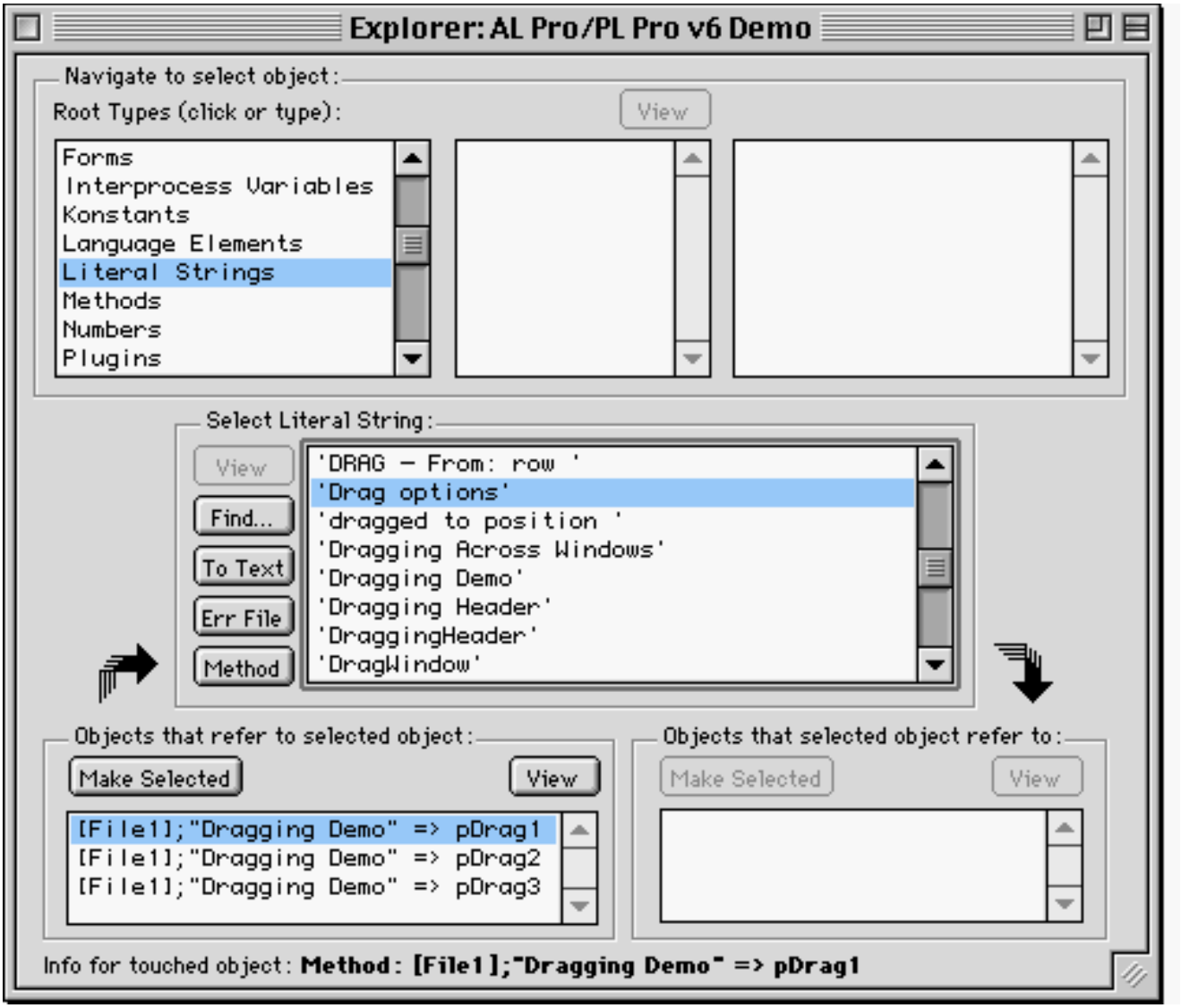
- Do not try to delete FI4D (table/file) objects. You will be almost guaranteed to cause serious problems to your structure file.
- Do not delete your PA4D objects - 4D won't be able to open the structure, and SanityCheck won't be able to help you re-create your lost password info.
- Note that when SanityCheck is done deleting the object, and you run SanityCheck, some new warnings are generated. They are of type [311] and talk about ‘dead space’ occuring in your structure. This is due to the fact that although the object has been removed, the bitmap/freemap was not updated by SanityCheck. This is done so that in case there was bitmap/freemap damage, SanityCheck is not affected, and the deletion is clean. You can run 4D Tools:Compact to clean up the dead space in your bitmap/freemap.

How to Use the SanityCheck Explorer

This chapter tells you how to use the SanityCheck Explorer.

Understanding the Explorer

The Explorer is a powerful cross-reference interface into the contents of your 4D structure files. The concept is similar to that of 4D Insider.



Navigating this window is done in a left-to-right, top-to-bottom order. When you first open the Explorer window, only the upper left list has content. Clicking on an item in this list will fill either the list on the right (tables and forms), or the list in the middle of the window

(everything else).

Optimizing for Stack Space

This chapter describes how to use SanityCheck to optimize for Stack Space.

The Basics About Optimization and the Stack

Your application will run faster if you do the following:

- Reduce the number of item 123's that are reported in your database.
- Reduce the number of item 107's that are reported in your database.

SanityCheck provides these two items specifically to help you optimize your stack space, which usually will make your application execute faster and in less memory. The following is a more in depth discussion of stack space and stack space usage.

What is Stack Space?

In general, a stack space is very much like a dead-end railroad track: the last car you push down the dead-end is always the first car you must retrieve. Similarly, your global procedures, when called at runtime, create a stack. The last procedure called is always the first procedure to be pulled off the end of the stack. So your procedures are stacked like railroad cars and as the current procedure completes, the next-to-last procedure can continue execution.

Local Variables and the Stack

When a local variable is defined in a structure file, space for the storage of that variables needs to be reserved someplace in memory. In 4D, as in many other environments, local variable space is allocated on the stack.

Your local variables are conveniently pushed onto the stack right after the procedure that contains the local variables is pushed onto the stack. So when your current procedure is finished, the local variables are pulled off the stack, and then the procedure is pulled off the stack. It is a perfect place to store your local variables since they only take up space in memory when the procedure is actually running, and also so that each instance of a procedure can have its own set of local variables; hence, recursion can work.

So what's the problem?

As more is pushed onto the stack, more work has to be done, and more memory is temporarily being used. A good practice is to minimize the amount of stack space used by your procedures, for better performance and less memory usage.

How to Use SanityCheck to Optimize Your Database

SanityCheck offers two major ways to help you optimize for stack space, and hence save memory and boost performance.

Item 123: Explicitly declared but unused locals

Item 123 is generated when you have explicitly defined a local variable via a compiler directive (**C_INTEGER**, **C_BOOLEAN**, etc.), but never use that local variable in your procedure. Neither the compiler nor the interpreter recognizes that the variable is defined but never used, so the stack space is allocated anyway. This is not going to cause any slowdown or loss of memory for a few extra integers that are defined but never used, but may be a concern if you define **C_STRING**(255;$string) a few times and don't need to. The one call to **C_STRING** shown allocates 255 bytes on the stack. If you do this 4 times, you have allocated nearly 1 K of memory on the stack. These types of local variables initializations can add up quickly.

When you get an item 123, simply locate the declaration for the local variable, and delete it.

Item 107: Local Vars that are undeclared

Item 107 is generated when a local variable is used before it is explicitly defined. This means that you accessed the variable before it appears in a compiler directive (**C_INTEGER**, **C_STRING**, etc.). Many developers have noticed that the compiler and interpreter, when typing a variable, will always type the variable to the "loosest" constraint possible. For example, the assignment "$i:=1", the local variable "$i" will be typed as a real, which is the least constraining type of number. As you might suspect, a real variable takes up more space than an integer, and you can begin saving stack space by watching how every local variable is allocated. SanityCheck lets you know which ones you missed.

You should keep in mind that local variables used as a loop counter (e.g., "**For** ($i;1;100)") will run faster than if the loop variable is typed as a longint (e.g., "**C_LONGINT**($i)") than if they are defined as real,

which would be the default.

NOTE: 4D Compiler v2 allows you to change the default types, but you should be very careful before you change this on an existing project, as you may accidently change the behavior of procedures that assume their local variables have been defined as real.

Optimizing Layout Size

This chapter describes how to use SanityCheck to optimize your layout size.

Why is Layout Size Important?

The size of a layout is directly related to how long the layout takes to load when being displayed. The layout size is determined by the objects on the layout.

What is stored in a layout?

1 All pages of a layout are stored in one block on disk.
2 All pictures, buttons, etc., are stored in the layout on disk.
3 Layout procedure and scripts are all stored elsewhere.

Picture sizes.

The largest pieces of a layout, in most cases, are the pictures. Here are the some things that affect picture size:

1 Actual rectangle of the picture
2 Color depth of the pixels.
3 Size of color table for the picture.

How to Identify Layouts with Excessive Size

Reported Items 119 and 120.

Items 119 and 120 are the two items that allow you to fine-tune the size of your layouts, which in turn make your layouts load faster, and take up less space on disk.

These items are governed by two tolerances, set in the Tolerances dialog. You may change these values to fit your specific needs.

The Factory Defaults will catch most severe layout size and picture size problems.

Putting it all Together

You want to keep your picture size down by being careful on exactly how you create your pictures and keeping track of which ones need

to be in color and which do not (see the next section for more information).

You want to keep your layout size down by watching your picture sizes and watching how many pages are in each layout. SanityCheck doesn't directly report the number of pages, but when SanityCheck notes a large layout, it may be simply because it has many pages. By not having as many pages, the layout will load faster.

How to reduce the sizes of small pictures.

Many people look at a 10 pixel by 30 pixel black and white picture and never imagine that it takes up 2K of disk space. The usual problem? Although it looks black and white, it's really color, and may have a very large color table inside of the picture.

The solution in most cases is to use a screen capture utility, such as SnapJot or Capture. Many developers turn their screen to black-and-white, take a snapshot of the area of the screen of interest, and paste the captured area of the screen back into the layout. This process removes any doubt as to the structure of the picture: it will be a bitmap with no color information.

Layouts with a lot of objects.

Although SanityCheck does not allow you to set a tolerances on the number of objects in a layout, you can eyeball your larger layouts and consider replacing large numbers of static text objects and static lines on the screen with one large picture. In many cases, the large picture (especially if it is in black and white) can load faster than the many small objects.

Many developers have found that replacing all text and lines with pictures can dramatically speed up the loading and displaying of a layout. Beware, though, that this approach leads to maintenance issues as you must change a large picture every time a piece of text on screen changes.

Common Questions

This chapter has common questions and answers that many people have asked about SanityCheck.

When does SanityCheck open/close the structure?

SanityCheck doesn't actually open the structure file until you begin to scan the structure file. As far as the Macintosh operating system is concerned, the file is only open for the short period of time that the scan is actually occurring. As soon as the scan is finished, SanityCheck closes the structure file. Since SanityCheck keeps the structure file open for such a short period of time, and since SanityCheck only opens the structure file "read-only", you can keep SanityCheck open and ready to run while you work.

How does SanityCheck pickup my structure changes?

SanityCheck re-scans the entire structure file every time you press the Start button or begin a Find operation. For SanityCheck to pick up your changes, you must first save any changes that you have made in the structure file. For example, the quickest way to save a procedure is to choose the Save command in the File menu, or press command-s.

If you have changed several layouts and procedures, the quickest way to save the changes is to enter the User or Runtime environment. 4D will automatically save any changes you've made.

Types of Items Reported by SanityCheck

SanityCheck identifies and reports over 200 different items when scanning a database.These items are categorized by severity, as shown Table 1 on page 45.

Table 1: Types of Items Reported by Sanity Check

Warning	A warning is an item that you may wish to investigate, but appears for your information only. For example, if your tolerances are set to warn when any layout exceeds 10K in size, then you will get a “warning” about any layouts that exceed 10K.
Error	An error is a situation that you should consider taking action to correct. For example, all syntax items are considered errors.
Fatal Error	A fatal error occurs when SanityCheck cannot continue processing the current object due to the encountered item. For example, if a layout is damaged, then SanityCheck will generate a fatal error telling you where the error exists. SanityCheck will immediately stop processing the layout, and continue processing the rest of the structure file. In some severe cases, SanityCheck cannot continue at all and will simply stop. Fatal errors will always appear during find operations, since they cause the processing to skip parts of your structure file where results of your find operation may exist.

Can you recover a password?

No. Committed Software is doing the 4D Community a service by verifying that you are a designer of a database before allowing you designer type access.

Due to legal and ethical reasons, Committed Software will not help any developer recover his or her password. Period. Don’t even bother asking. We’ll say no. I promise. No.

Did ACI help with this product?

No. SanityCheck was written from scratch without ACI involvement, acknowledgment, or help.

Using 4D Tools:Compact

This chapter describes how to effectively use 4D Tools:Compact, and why SanityCheck requests that you perform this lengthy operation.

Using 4D Tools:Compact Efficiently

4D Tools has many functions for managing your 4D project. The one function that SanityCheck often requests you to perform is the Compact feature, which compacts both the structure file and the data file. However, when SanityCheck mentions running 4D Tools:Compact, only the structure file needs to be compacted.

To Use 4D Tools:Compact v 3.2 or Later

These versions of 4D Tools allows you to only compact the structure file.

1. Drag your structure onto 4D Tools.
2. Choose the Compact in the Utilities menu.
3. Type the new structure name
4. Click the Save button.
   After the structure is compacted, 4D Tools will prompt you for a new data file name.
5. Click the Cancel button.

To Use 4D Tools:Compact Version 3.1 or Earlier:

For these versions of 4D, you need to create an empty data file so that the compaction doesn't take all day and all night. Here is a brief description of how to accomplish this:

1. Double click on your structure file and then hold down the option key.
   4D will request you to open the data file.
2. Click the New button to create a new, empty data file.
3. Quit 4D.
4. Open the structure file to verify that the proper (new, empty) data file is opened.
5. Launch 4D Tools and open the structure file from 4D tools.
6. Choose Compact in the Utilities menu.

When 4D Tools is done, you will have a new structure file. You will want to save the old structure file until you are confident that the new structure file is operational.

HINT: Many developers keep around a current “empty” data file specifically for this compaction purpose, as it can take some time to create the empty data file.

What does the compaction operation do?

The compaction operation was designed to get rid of a common problem that occurs in many databases (the structure file can be called a database). The problem is fragmentation of the structure file. As you create, modify and delete layouts, objects and procedures, small pieces of unused space are created in the structure file. The compaction process lines up all of the objects (layouts, procedures, etc.) in the structure file so that they take up as little space as possible. As you design some more, more empty space is created and compaction will again shrink the size of your structure file.

Why does SanityCheck want me to Compact?

Since the compaction process rebuilds and rewrites every object within the structure file, many problems that may creep into the structure file are solved as a side affect of the compaction process. Any damage that occurs within the Object Map is a prime candidate for running 4D Tools:Compact.

While developing SanityCheck, we have learned a lot about how the different 4D related tools affect the structure file. We specifically recommend using 4D Tools:Compact in situations we believe it has been shown to help in the past.

However, it can never hurt to run 4D Tools:Compact either periodically or when you suspect problems with your structure file.

Recovery Tips

This chapter describes a quick check list of things to think about when you suspect a damaged structure file.

Don't Panic!

The first thing you need to do if you suspect damage, is to logically think about what could be causing the problem. However, before you do anything, you should follow these instructions:

1. Do not copy *anything* to the disk that has your structure file on it.
2. Copy the structure file that you think is damaged to another disk.
3. Preferably disconnect the disk where the suspected damage occurred and set it aside.

The reason that you should do this is that if you had a SCSI or disk related problem, any modifications made to the source disk may damage your structure file further.

What should I try?

1. Restart with no inits installed (System 7 users hold down the shift key while restarting).
2. Run all the 4D application tools you have that will open a structure file and see how each one reacts. These tools include: *SanityCheck*, *4D Tools:Compact*, *4D Insider*, *4D XREF*, *4D Compiler*, and *4th Dimension*.
3. If you have a Proc.Ext file, try throwing it away and recreating it from scratch.
4. If you do not have a Proc.ext, use 4D External Mover (and for those externals with their own installer, be sure to use that installer, *not* External Mover) to remove all externals from your project and reinstall them all from scratch. Resedit savvy users may use a more direct method: open the resource fork and delete everything, then reinstall. Only 2.2.3 users need to worry about the EN4D resource, which are the 4D lists. **IF YOU DON'T KNOW WHAT YOU'RE DOING, DON'T DO IT.**
5. Check how much memory is allocated to 4D.
6. Using ResEdit, perform a "Verify" on the resource fork of 4D, 4D Compiler, 4D Insider, etc.

And if that doesn't work…

If you are convinced that your structure file is damaged, then you should recover from a backup. You should make backups often, and

keep enough in case a problem creeps in and it takes you some time to detect.

If you have had a hardware crash, there are several hardware recovery services available that will do a good job of getting the file off of the disk, even when something like Norton Utilities fails. Be prepared to pay for this, and be aware that in most cases, you are not going to be very successful, unless the disk crash didn't affect the portion of the disk where your structure file existed. Check the back of MacWeek or MacWorld for these vendors.

Lastly, Committed Software is still in the business of recovering structure files that have been damaged beyond the hope of any tool available. This work is grueling and you should be prepared to pay a large chunk of change for making us stare at disk blocks all day long. Call for our rates.

Comparison Functionality

What does SanityCheck compare?

SanityCheck compares all Layouts, Layout Procedures, Scripts and Procedures. No comparison is done on other areas, such as the data dictionary, menus, balloon help, etc. Future releases of SanityCheck may implement comparison of these objects as well.

Do not assume that two structure files are equal if SanityCheck doesn't find a difference. Since SanityCheck doesn't compare all objects, there may be differences between the structures files, even if SanityCheck doesn't note them.

Comparison Caveats:

1. Note that the comparison is binary in nature. This means that even reading in an object, making no changes but forcing 4D to save the object may lead to a binary difference between the original object and the newly "changed" object. This is due to extra space that is saved with the object. This extra space is random in nature.
2. Due to Item 1 above, any object moved by Insider will show up as changed.
3. Layout procedures/scripts are only scanned if its parent layout passes comparison. Therefore, if a layout is different, assume the layout procedure and scripts may be different as well.
4. Sometimes changing a script will force a change in the parent layout. Be aware that when a layout is tagged as "changed" it may only be a script that has been modified, which has caused the layout to be re-written.

To Compare Two Structure Files

1. Open two structure files within SanityCheck.
2. Choose the Compare Structure Files command in the Special menu.
   If you do not have exactly two structure files open, the Compare Structure Files option is still present, but directs you to open exactly two structure files open.

How does SanityCheck perform comparison?

SanityCheck does a binary comparison between objects. Each byte of one object must equal the same byte in the compared object. Note that two objects that are functionally equivalent may not be

equal when compared in a binary fashion. This is due for two reasons:

1. Layouts and Procedures have “dead space” in them often, which will get filled with random information. For example, each layout object has a name. There is space reserved for the name of the object in the layout, but if the name is short, the extra space not used by the name is “dead space” and is filled with whatever happens to be in memory at the time (i.e., it’s random). So when the layout or procedure gets saved back to disk, different information may be stored in the layout or procedure.
2. 4D Insider puts extra, unused information on the end of every line of every procedure or script that is touched by Insider. This modification can render a binary comparison useless for Insider manipulated files.

Why a binary comparison?

Although the author of SanityCheck is quite knowledgable about the internals of the structure file, it is possible that there are areas of the structure file, or procedure and layouts in particular, that have not been fully discovered. It is therefore possible that if SanityCheck offered a functional equivalence comparison test, that the resulting comparison would not be thorough. However, a binary comparison will always work, but will tag differences between objects that are functionally equivalent. We decided that it was better to err on the side of showing more differences than attempting to show functional differences. You, as developer, need to be an active part of the comparison process — determining which comparison differences are in reality functionally equivalent.

Why does SanityCheck say a layout or procedure has changed when it hasn’t?

SanityCheck performs what is called a binary comparison between objects (layouts, procedures, etc.). Even if all you do is open a layout and save it and quit 4D, this may trigger SanityCheck to see differences due to the save operation. Please read the section “How does SanityCheck perform comparison?” on page 50 and “Why a binary comparison?” on page 51 for more information.

Why don’t layout procedures and scripts always show up as modified?

If a layout has changed, SanityCheck cannot compare the layout procedure and scripts associated with the layout. Often times, changing a script will force 4D to modify the layout. Be aware that all

layout changes mean that script changes may also have occurred.

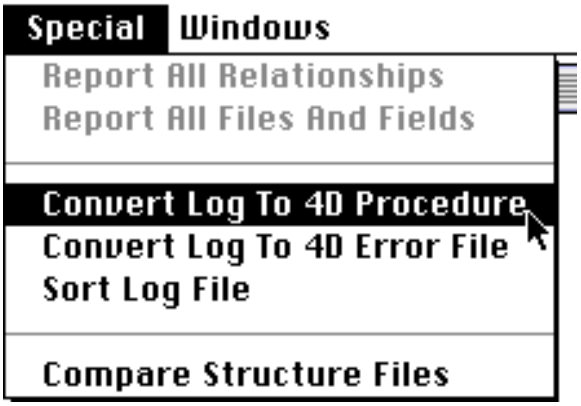
The detailed reason why this happens has to do with how layouts are stored. The object in the layout are stored in the order that they are drawn on screen. The only semi-unique identifier for most objects with scripts is the name of the object (a button's variable name might be "bOK"). Note that this is not a unique identifier, since several objects can have the same name on the same page of the same layout. Therefore, SanityCheck has no unique way of identifying an object on the layout and therefore can't identify its related object on the layout being compared, so the comparison cannot be done. Future versions of SanityCheck and/or 4D may allow this restriction to go away.

When does 4D save a layout or procedure?

4D automatically saves a layout or procedure when you close it, if any modifications have been made. In a procedure, you can cause a modification by simply opening the procedure and pressing command-enter. Of course, selecting "Save" or pressing command-S will save the disk version of the layout or procedure.

Log Conversions

Convert a Log to 4D Procedure



This report can only be run when the frontmost window in Sanity-Check is a log file generated by a SanityCheck scan.

This conversion will take global procedure and layout error items and generate a procedure suitable for pasting into a 4D procedure. Once the procedure is pasted into 4D, you can use it as a “jumping off point” for exploring the items that SanityCheck has noted.

To Convert a Log to a 4D Procedure

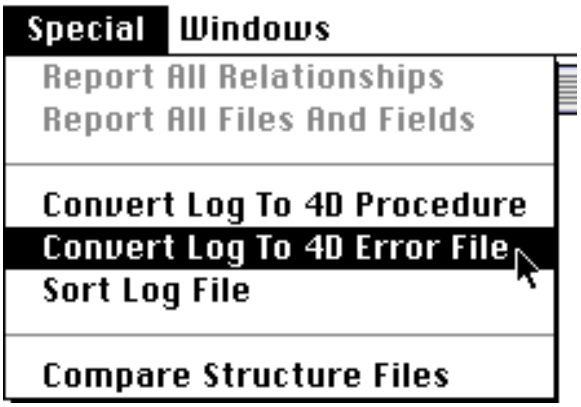
1. Make sure a log window is the foremost window.
2. Choose the Convert Log to 4D Procedure command in the Special menu.
   The log is now converted to a 4D Procedure, and is placed in a new window.

To Use a Log which has been Converted to a 4D Procedure

1. Select the text in the SanityCheck window containing the converted log.
2. Copy the text to the clipboard.
3. Open the database using 4th Dimension.
4. Enter the Design Environment.
5. Create a new procedure, or open an existing procedure.
6. Paste the converted log into the procedure.
7. Double click on a procedure name and press command-P.
   The procedure will open automatically.
8. Double click on a layout name and press command-L.
   The layout will open automatically.

All script items are put into the 4D procedure as the layout that they are associated with because there isn't a way to ask 4D to open a specific script within a layout.

Convert a Log to a 4D Error file.



This report can only be run when the foremost window in Sanity-Check is a log file generated by a SanityCheck scan.

This conversion will take global procedure items and generate an error (.err) file in the same directory as your structure file. Note that the old .err file (if present) is destroyed. The .err file's format is compatible with the error file generated by the 4D Compiler.

When 4D is launched, it reads this .err file, and puts additional menu items in the Use menu in the Design Environment. You can use the Next Compiler Error menu item to see the errors that SanityCheck has generated.

Be aware that SanityCheck cannot generate the error file when 4D is currently using it, so you may need to switch to user mode or choose the Stop browsing error file command in the Use menu to allow SanityCheck to write out the .err file.

Be aware that some errors that SanityCheck generates have no "line number" associated with them. In these cases, 4th Dimension will jump to the end of the procedure.

Be aware that some versions of 4D cannot select the last line of the procedure. If an error occurs in the last line, sometimes it will not be highlighted.

There is no facility to put layout or menu or data dictionary items into the .err file, so it is limited to scripts and procedures.

McCabe Complexity

What is McCabe Complexity?

Wouldn't it be great if you could have a way of automatically knowing where, in all of your code, you are most likely to have bugs?

McCabe Complexity is a very simple algorithm that can be applied to any programming language to quickly identify areas of "complexity" in your code. It's for sure that any complex area of code is more likely to have bugs that those that are less complex.

The concept is very simple: simply add up all of the "paths" your method can take (For example, each "if" statement is a fork in the code, which makes a new path). The resulting number is the complexity. Although this sounds almost too simple to be useful, you may be surprised how it can be used to very quickly identify complicated routines.

The algorithm is also very simple. You simply add up the decision points:

1. Start with complexity of 1 (the default single path through method)

2. For each "If", "While", "For", "Repeat" and ":"(as part of a case statement), add 1 to the complexity (it's a new potential path the code handles).

3. For each "AND" or "OR" case in an "If", "While", "For", "Repeat" and "Colon" also addd 1 to the complexity (each OR or AND is a different path through the code, depentant on the result of that comparison).

That's it. Add all those things up and you get a number. There are many different things you can then do with this number (divide it by the number of lines, for example), or many other things to try to refine it.

SanityCheck only computes this number as it's the simplest to understand and does the job of identifing complexity relatively effectively.

Caveats with 4D Code

Some programming styles within 4D make heavy usage of the

CASE OF statement, and if you have long methods of case statements, say for error code string resolution, you may end up with a “high complexity” for this routine. That’s OK. The goal of complexity is to just help identify those areas that may be complex. Use the McCabe complexity to identify those areas and then consider whether you want to restructure your code.

How do I make my code less complex?

The simple answer is to factor your code. By factoring, you create several methods to accomplish the same task as the original method. Usually, the original method becomes a “ring-master” calling the new methods to accomplish some of the “sub-tasks” of the original, large method.

By choosing good places to create new methods you will find that the code is much more readable (less complex) and easier to maintain.

There are many books on the subject of complexity and factoring code. Code Complete (Steve McConnell, Microsoft press) is a great place to get started along the path of making your code less complex, as there are great discussions on complexity

.

Variable Span / Live Time

What is Variable Span?

Another great way of detecting complexity in a method is to count how many lines of code exist between each reference to a variable.

For example, examine the Example 1 given below:

Example 1:

```
$countFiles:=1;
While (GetNextFile($name))
{
  ...do something...
  ...do something..
  ...do something...
  ...do something...
  ...do something...
  $countFiles:=$countFiles+1
}
```

The variable span for $countFiles is 8. You may also notice that there's no particular reason for putting the bump for $countFiles at the end of the whle loop.

If we restructure this to be:

Example 2:

```
$countFiles:=1;
While (GetNextFile($name))
{
  $countFiles:=$countFiles+1
  ...do something...
  ...do something..
  ...do something...
  ...do something...
  ...do something...
}
```

Then the span drops down to 3.

But what we can see here is the direct relationship between the numbers generated (8->3) and what that does to the readability of the code.

The $countFiles variable is now packed together at one place in the method and it's therefore much easier to understand the full usage

of $countFiles.

What is Live Time?

Live time tells you the span from the first reference of the variable to the last reference of the variable. In other words, it tells you how long the varaible is “live” in the method.

As in the above examples, it can easily be seen why it is advantageous to reduce this number, as it will tend to “pack” your variables together and make it easier to read (and maintain) your code.

A warning and simple advice

When you are simply restructuring your code to reduce complexity or VariableSpan or Live Time, you need to be very careful not to change the functionality of the code.

This is a great reason to use your code metric tool (SanityCheck in this case) during development so that you catch such issues early and work around them during development.

Reported Items

1 Cannot read structure's header.

Detailed Description: This item will occur when the header for your whole structure file cannot be read into memory. When SanityCheck begins to scan your structure file, the very first thing SanityCheck does is read in the header of the file. If it cannot read that header, then this item is reported.

Actions: Usually, there is nothing you can do except recover from a backup.

2 Structure file type is bad. This may be a newer version of 4th Dimension.

Detailed Description: Most likely, this structure file was created by a newer version of 4D than is supported by SanityCheck. This version of SanityCheck supports all version 2.X and 3.X structure files, up to and including 3.1.1. This may also be generated if the structure file has been damaged.

Actions: If the structure file was last modified with a version of 4D within the range described above, then you may have a damaged structure file. For example, if SanityCheck has always run cleanly on your structure file, and you've not changed your version of 4D, then your structure may be damaged. If it is, there is nothing you can do except recover from a backup. If this is a more recent version of 4D, contact Committed Softwar for an upgrade to SanityCheck.

3 This file is too small to be a 4th Dimension structure file.

Detailed Description: There is a minimum file size that is necessary for 4D to open a structure file. In other words, if the file size is less than a certain amount, then the header of the file is truncated, and therefore cannot be a valid structure file.

Actions: You will need to recover from a backup. This probably will only occur if you are having serious disk problems, so you may wish to run some disk utilities on your disk to figure out what is going wrong.

4 The object map is described as beginning after the end of the file.

Detailed Description: SanityCheck needs to read in the Object Map to be able to find all of the Structure Objects within your structure file. The structure file contains information about where to find this Object Map. This item occurs when the description of the location of the Object Map is that it begins after the end of the file. This is a very serious problem.

Actions: You will need to recover from a backup.

5 The Object Map continues past the end of the file.

Detailed Description: SanityCheck needs to read in the Object Map to be able to find all of the Structure Objects within your structure file. The structure file contains information about where to find this Object Map. This item occurs when the description of the location of the Object Map together with the description of the size of the Object Map, places the object map outside the range of the file on disk.

Actions: You will need to recover from a backup.

6 An Internal error has occurred.

Detailed Description: SanityCheck has had an internal error that it cannot recover from.

Actions: Try restarting your Macintosh and running SanityCheck again. Try restarting with no INITs (system 7, just hold down the SHIFT key while restarting your Macintosh) and try again. If this fails, reinstall SanityCheck from the master disk (it may have been damaged). If you still get this error, contact Committed Software for instructions.

7 Object Map describes object begins after then end of the file.

Detailed Description: The Object Map describes the location of an object as beginning beyond the end of the file.

Actions: You will need to recover from a backup.

8 Object Map describes object as continuing past the end of the file.

Detailed Description: The Object Map describes the length of an object as continuing beyond the end of the file. See glossary for information about the "Internal Object ID".

Actions: You will need to recover from a backup.

9 The Object Map looks to be completely empty.

Detailed Description: The Object Map has no information in it. Normally, the Object Map contains several “must have” objects in order for the structure file to be properly formed.

Actions: You will need to recover from a backup.

10 End of the Object Map while still parsing. Object Map must not be intact.

Detailed Description: While reading the Object Map, SanityCheck ran off the end of space allocated for the Object Map.

Actions: You will need to recover from a backup.

11 There are too many object types for this program to function.

Detailed Description: SanityCheck can only handle 64 different types of objects. This far exceeds the number of object types ever encountered within the 4th Dimension structure file. If this item occurs, there is most likely a problem with the structure file.

Actions: If you have just upgraded to a new version of SanityCheck in order to be able to run on a new version of 4D's structure file, then you should contact Committed Software. It may be that there is a problem with SanityCheck running on the new version. Otherwise, You will need to recover from a backup.

12 Object Map defines an object which exists outside of the file (<1>).

Detailed Description: This item will occur just before an object is about to be read if the object either begins past the end of the file, or continues past the end of the file. This check is actually done twice (see 7/8 above). The 7/8 is done during start up during a quick-check scenario. This check is done just before the object is actually read into memory.

Actions: You will need to recover from a backup.

13 Global Procedure List has a malformed header.

Detailed Description: The list of global procedures has been successfully read into memory by SanityCheck, but appears to be damaged. The global procedure list of the names of all global procedures that exist in the structure file.

Actions: Try running 4D Tools:Compact. If that doesn’t help, recover from a

backup.

14 Global procedure '<1>' has no associated data (i.e., it doesn't really exist).

Detailed Description: The global procedure named in <1> cannot be found on disk.

Actions: You will need to recover this procedure from a backup. Try deleting the procedure and re-creating it. If you still have problems, try running 4D Tools:Compact. If you continue to have problems, you may need to recover the entire structure from a backup.

15 Global procedure name is too long: '<1>'.

Detailed Description: The global procedure has a name that exceeds the maximum name length allowed. The maximum is 15 characters. The name shown is a truncated version of the name found in the file.

Actions: Open 4D and locate this procedure. Try retyping the name of the procedure. If you continue getting this item, try deleting the procedure and re-creating it. If you still have problems, try running 4D Tools:Compact. If you continue to have problems, you may need to recover the whole structure from a backup.

16 Procedure/script has a malformed header.

Detailed Description: The procedure or script has been successfully read into memory, but appears to be damaged.

Actions: Delete the procedure or script and re-enter it. You may use “Clear Scripts” to delete the script, or just delete the layout object that contains the script. Try running 4D Tools:Compact. If you continue to have problems, you may need to recover the entire structure from a backup.

17 File Name List has a malformed header.

Detailed Description: The list of file names has been successfully read into memory, but appears to be damaged.

Actions: Try running 4D Tools:Compact. If you continue to receive this item, then you will need to recover from a backup.

18 File number <1> has a bad file name.

Detailed Description: The name stored for the file is too long. The maximum number of characters is 15.

Actions: Try renaming the file. Try running 4D Tools:Compact. If these two fail, recover from a backup.

19 Too many files (count=<1>) for SanityCheck to handle.

Detailed Description: SanityCheck can only handle 512 files. The count given is the number of files that the structure file claims to have.

Actions: If you do have more than 512 files, contact Committed Software for an upgrade of SanityCheck. Try running 4D Tools:Compact. If you continue to have this error, recover from a backup.

20 A field list object has a malformed header.

Detailed Description: A field list object describes all the layouts for a particular file. This item occurs when the field list is read into memory, but appears to be damaged.

Actions: Try renaming all of the layouts for the file. Try running 4D Tools:Com-

pact. If you continue to have this error, recover from a backup.

21 Object Map problem. Object '<1>' claims to occupy the same space as '<2>'.

Detailed Description: The Object Map describes where each object exists on disk, and how long each object is in size. Using this information, SanityCheck verifies that no two objects overlap. If they do overlap, this item is generated. Internal Object IDs are used to describe the objects.

Actions: Run 4D Tools:Compact. After doing this, rerun SanityCheck, and you may find that you now have one (or two) objects that are damaged. You will then need to delete one or both of them and recreate them.

22 Unexpected end of procedure.

Detailed Description: This item occurs when SanityCheck cannot continue processing a procedure because the space allocated for the procedure has all been parsed, yet SanityCheck is expecting more information.

Actions: Run 4D Tools:Compact. If this doesn't help, delete the procedure and recreate it from scratch.

23 Invalid math token (<1>) on line <2>.

Detailed Description: This item occurs when SanityCheck encounters a Token that defines a mathematical operator exists in the procedure, but the operator is not a valid operator. This item will appear if the procedure is damaged, or if a new operator has been added to the 4D language that SanityCheck doesn't yet recognize.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If the operator that is in the line appears to be OK, and has just been added to the 4D language, then you should contact Committed Software for an upgrade. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

24 Invalid 4D function (<1>) on line <2>.

Detailed Description: This item occurs when SanityCheck encounters a 4D Function within your procedure that is not valid. This item will appear if the procedure is damaged, or if a new function has been added to the 4D language that SanityCheck doesn't yet recognize.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If the 4D function that is in the line appears to be OK, and has just been added to the 4D language, then you should contact Committed Software for an upgrade. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

25 Invalid 4D constant type (<1>) on line <2>.

Detailed Description: This item occurs when SanityCheck encounters a constant whose type is not recognized. A constant is a number, or a piece of text. There are several types of constants supported within 4D. This item will appear if the procedure is damaged, or if a new constant type has been added to the 4D language that SanityCheck doesn't yet recognize.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If the constant that is in the line appears to be OK, and has just been added to the 4D language, then you should contact Committed Software for an upgrade. If SanityCheck still

encounters the item, delete the procedure and recreate it from scratch.

26 Procedure references field with too many subfields (<1>) on line <2>. Max is 5.

Detailed Description: This item occurs when SanityCheck encounters a subfile reference that has too many children in it. SanityCheck only allows 5 subchildren to exist, as in: “[MyFile]Child1'Child2'Child3'Child4'Child5”. It assumes that any more subfields than that are erroneous.

Actions: If you really want to have more than 5 subchildren, contact Committed Software for an upgrade and a free “how to design your database” lecture. Otherwise, if you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

27 Procedure referencing file/field that doesn't exist: <1>. (line: <2>).

Detailed Description: This item occurs when SanityCheck encounters a file, field, or subfile combination that does not exist in the structure file. All file and field references are stored numerically inside of a 4D procedure. When you type “[File1]” it gets translated and stored on disk as “<FileToken>,1”. When SanityCheck parses this token, it verifies that a file exists with the number 1.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

28 Procedure has a var with an invalid name (name length: <1>, line: <2>).

29 Proc calls a procedure with an invalid name (name length: <1>, line: <2>).

30 Procedure calls an external with an invalid name (name length: <1>, line: <2>).

31 Proc has comment which is too long (comment length: <1>, line: <2>).

32 Proc has local var with an invalid name (name length: <1>, line: <2>).

33 Proc has interprocess var with an invalid name (name len: <1>, line: <2>).

Detailed Description: These items occur when SanityCheck encounters a variable, procedure, external, etc., whose name is an invalid length. The valid range for variables, local vars, and interprocess vars is 1 to 11, inclusive. The valid range for procedures and externals is 1 to 16, inclusive. The valid range for comments is 0 to 80, inclusive.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

34 Calls procedure '<1>' which doesn't exist (line: <2>).

Detailed Description: This item occurs when SanityCheck encounters a call to a procedure that doesn’t exist.

Actions: If the procedure doesn’t exist, then create it or delete the reference to it in the calling procedure. Otherwise, if you can, inspect the line. If the line looks correct (and the procedure does indeed exist), try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

35 Procedure has an unknown token (<1>) on line <2>.

Detailed Description: This item occurs when SanityCheck encounters a Token that SanityCheck does not recognize. This item will appear if the procedure is damaged, or if a new token has been added to the 4D language that SanityCheck doesn't yet recognize.

Actions: If you can, inspect the line. If the line looks correct, try retyping it. If SanityCheck still reports this item, try retyping the whole procedure. If the line contains code that has only just been added to the 4D language, then you should Software for an upgrade. If SanityCheck still encounters the item, delete the procedure and recreate it from scratch.

37 Too many fields in file (file num: <1>).

Detailed Description: SanityCheck only allows up to 512 fields in each file. This item is generated when SanityCheck encounters more than 512 fields in a particular file.

Actions: If you are using a version of 4D that allows more than 512 fields, and you really want more than 512 fields in your file, contact Committed Software for an upgrade. Otherwise, there may be a problem with your structure file. Try running 4D Tools:Compact. Try renaming the file and all its fields (this forces a re-write of the file to disk) and then running 4D Tools:Compact.

38 Unexpected end of object encountered in file '<1>'.

Detailed Description: This item is generated when SanityCheck is parsing the description of a 4D File and expects more information to exist than is there.

Actions: Recover from a backup.

39 Ascii field named '[<1>]:<2>' has an invalid length.

Detailed Description: This item is generated when SanityCheck encounters an ascii field that has a length that is not in the valid range of 2 to 80, inclusive. This item occurs while parsing the data dictionary (the Structure view in Design mode).

Actions: If you can, open the database with 4D, and inspect the file/field definition for this file in the Structure view in design mode. Retype the size of this ascii field and save the structure file. Re-run SanityCheck. If the item is still generated, try change the name of every field in the file (this forces the file to be re-written). If you still have problems, you may need to recover from a backup.

40 More files exist than are reported mentioned. Failed at file '<1>' (reported: <2>).

Detailed Description: This item is generated when SanityCheck encounters a file whose file number is outside the range of file numbers that have been defined for this structure. File numbers will always start at 1, and increase monotonically to the maximum number of files in your structure.

Actions: Check how many files your structure is supposed to have against the number that is reported in the item by SanityCheck. Try renaming the file that is mentioned (this forces that file description to be rewritten to disk). If SanityCheck still reports the item, try renaming all of the files in your database and then running 4D Tools:Compact. If you still get this item, you need to recover from a backup.

41 Field named '[<1>]<2>' has an undefined type.

Detailed Description: All fields have a type (Real, Boolean, etc.). SanityCheck

verifies that all fields have a valid type. SanityCheck reports this item when an undefined type is found for the given file/field.

Actions: If you are using a new version of 4D, and there is a new type added to the 4D database, then you need to contact Committed Software for an upgrade. Otherwise, you should open the structure file and inspect the field definition for the named field. Reenter the field definition, and change the name of the field (this forces a rewrite of the field description). You may need to run your structure through 4D Tools:Compact. If SanityCheck still reports this item, then retype the name of the file. If SanityCheck still reports the item, you need to recover from a backup

42 Cannot read header of list of Balloon Help items.

Detailed Description: This item is generated when the list of balloon help items cannot be read.

Actions: Try running 4D Tools:Compact. Try editing your balloon help items and changing the names of some of the items. If SanityCheck still complains, recover from a backup.

43 Balloon help has a poorly formed name (name = '<1>', given length = <2>).

Detailed Description: This item is generated when a balloon help name is too long. The name given is truncated to 15 chars (the max).

Actions: Edit your balloon help items and change the name of the balloon help item.

44 Balloon help has a poorly formed name (name = '<1>', given length = <2>).

Detailed Description: This item is generated when a balloon help name is too long. The name given is truncated to 15 chars (the max).

Actions: Edit your balloon help items and change the name of the balloon help item.

45 Cannot find text for balloon help named '<1>'.

Detailed Description: This item is generated when a the text for a particular balloon help object cannot be found in the Object Map. Essentially, the balloon help exists, but the object that contains the text for the balloon help item is missing.

Actions: Edit the balloon help and re-enter the text for the balloon help. Run 4D Tools:Compact.

46 The text for the balloon help item named '<1>' has a malformed header.

Detailed Description: This item is generated when a the text for a particular balloon help object has been successfully read into memory, but appears to be damaged.

Actions: Edit the balloon help and re-enter the text for the balloon help. Run 4D Tools:Compact.

47 The text for the balloon help item named '<1>' is empty.

Detailed Description: This item is generated when a the text for a particular balloon help object has a zero length. This is not a problem at all, yet you may wish to address it, in case you really wanted some message to appear.

Actions: Edit the balloon help and enter the text for the balloon help.

48 A balloon help item has no name.

Detailed Description: This item is generated when a balloon help item has no name. This is not a problem at all, yet you may wish to address it for maintainability.

Actions: Edit the balloon help and add a name for all empty balloon help names.

49 Layout contains an unknown token (<1>).

Detailed Description: This item occurs when SanityCheck encounters a Token that SanityCheck does not recognize. This item will appear if the layout is damaged, or if a new token has been added to the 4D language that SanityCheck doesn't yet recognize.

Actions: If you can, inspect the layout. If the layout contains objects that have only just been added to the 4D language, then you should contact Committed Software for an upgrade. Otherwise, delete the layout and recreate it from scratch. Note: in version 3 databases, you can hold down the option key while selecting a layout from the layout list and the "thumb" is not drawn. This way, you can easily delete the layout.

50 Unexpected end of Layout.

Detailed Description: This item occurs when SanityCheck encounters the end of the layout while still expecting more information to follow.

Actions: Delete the layout and recreate it from scratch. Note: in version 3 databases, you can hold down the option key while selecting a layout from the layout list and the "thumb" is not drawn. This way, you can easily delete the layout.

51 Cannot find layout in layout list. Master layout list object may be damaged.

Detailed Description: This item occurs when SanityCheck cannot find the given layout in the master layout list.

Actions: Run 4D Tools:Compact. If the item still occurs, recover from a backup.

53 List of layouts for file '<1>' does not exist.

Detailed Description: This item occurs when SanityCheck cannot find the list of layouts for a particular file.

Actions: Go to the Structure view in Design mode and double-click on the name of the file. If you get a list of layouts, then change the names of these layouts. Run 4D Tools:Compact. If the item still occurs, recover from a backup.

54 List of layouts for file '<1>' has a malformed header.

Detailed Description: This item occurs when SanityCheck has successfully read in the list of layouts for the file but has found that the header of the list appears to be damaged.

Actions: Go to the Structure view in Design mode and double-click on the name of the file. If you get a list of layouts, then change the names of these layouts. Run 4D Tools:Compact. If the item still occurs, recover from a backup.

55 Unexpected end of object while parsing the list of layouts for file '<1>'.

Detailed Description: This item occurs when SanityCheck has successfully read in the list of layouts for the file but ran off the end of the list before successfully parsing the whole list.

Actions: Go to the Structure view in Design mode and double-click on the name

of the file. If you get a list of layouts, then change the names of these layouts. Run 4D Tools:Compact. If the item still occurs, recover from a backup.

56 Layout named '<2>' in file '<1>' does not have related layout data.

Detailed Description: This item occurs when SanityCheck is looking for a layout for a file but cannot find the data for the layout.

Actions: Go to the Structure view in Design mode and double-click on the name of the file. If you get a list of layouts, then change the name of layout named by the item. Run 4D Tools:Compact. If the item still occurs, recover from a backup.

57 Layout in file '<1>' has an invalid length (len=<2>).

Detailed Description: This item occurs when SanityCheck encounters a layout name whose length exceeds 15 chars.

Actions: Go to the Structure view in Design mode and double-click on the name of the file. If you get a list of layouts, then change the names of all layouts in the file. Run 4D Tools:Compact. If the item still occurs, recover from a backup.

58 Layout procedure referenced by layout '<2>' in file '<1>' does not exist.

Detailed Description: This item occurs when SanityCheck cannot find the layout referenced as the layout procedure for the given layout in the named file.

Actions: Try running 4D Tools:Compact. If the item still occurs, try deleting the layout and recreating it from scratch. If the item still occurs, recover from a backup.

59 File named '<1>' has too many layouts (n:<2>).

Detailed Description: SanityCheck only allows 1024 layouts to be in each file. This item is generated when more than 1024 layouts exist in a file.

Actions: If you wish to have more layouts in a particular file, contact Committed Software for an upgrade of SanityCheck. Otherwise, try running 4D Tools:Compact. If the item still occurs, try renaming the file, and all it's layouts. If the item still occurs, recover from a backup.

60 Object '<1>' in layout has a bogus name (too long: <2>).

Detailed Description: Since layout objects are variables, the names of layout objects can be at most 11 chars in length.

Actions: Locate the object on the layout and retype the object name. If the item still occurs, delete the object and recreate it. If the item still occurs, delete the layout and recreate it. Note: in version 3 databases, you can hold down the option key while selecting a layout from the layout list and the “thumb” is not drawn. This way, you can easily delete the layout.

63 Object in layout has a zero length name.

Detailed Description: It is acceptable for a layout variable to not have a name, however it is good programming practice to name each layout variable.

Actions: Locate the object on the layout and enter an object name.

64 Script cannot be found in Object Map to be verified.

Detailed Description: SanityCheck is attempting to locate the script for a given layout, and cannot find it in the Object Map.

Actions: Try running 4D Tools:Compact. If the item still appears, clear the script, and recreate it. If the item still appears, delete the object (button, variable, etc.)

that contains the script and recreate it from scratch.

65 Script cannot be read to be verified.

Detailed Description: SanityCheck has failed to read the script for a given layout.

Actions: Try running 4D Tools:Compact. If the item still appears, clear the script, and recreate it. If the item still appears, delete the object (button, variable, etc.) that contains the script and recreate it from scratch.

66 Balloon Help cannot be found.

Detailed Description: Every object (Variable, Button, etc.) within a layout contains information such as the location of the object, etc. Part of the information that is stored in the object is the internal ID of the balloon help for the object. SanityCheck generates this item when the balloon help that is referenced by the object doesn't exist in the structure. It appears that this situation occurs quite often in version 3 structure files. ACI has been made aware of the issue. However neither ACI, Committed Software, or any third party has been able to show that it causes any problems.

Actions: You can choose to ignore this item. If you would like to remove these items from your structure file, you may do so by opening the layout that has the object, double clicking on the object, clicking the Balloon Help button and then clicking the No Balloon button.

68 Password object has a malformed header.

Detailed Description: The password object has been successfully read into memory yet appears to be damaged.

Actions: Try opening the password dialog and modifying the passwords (which forces a re-write of the password object). If you still encounter this item, try 4D Tools:Compact. If you still encounter this item, recover from a backup.

69 Invalid password. Cannot continue without password for user '<1>'.

Detailed Description: SanityCheck requires that you know the Designer password to run SanityCheck. You will only have to enter the password the first time you scan the file.

Actions: Type in the correct password.

70 Unexpected end of object while parsing passwords.

Detailed Description: SanityCheck generates this item when it encounters the end of the password data and is still expecting more information.

Actions: Try opening the password dialog and modifying the passwords (which forces a re-write of the password objects). If you still encounter this item, try 4D Tools:Compact. If you still encounter this item, recover from a backup.

71 This program cannot recognize the version of 4th Dimension that it has opened.

Detailed Description: SanityCheck transparently supports all version 2 databases and all version 3 databases up to and including release 3.1.1. If ACI releases a new version of 4D SanityCheck may need to be upgraded.

Actions: If this structure file is from a new release of 4D, contact Committed Software for an upgrade for SanityCheck. Otherwise, run 4D Tools:Compact. If the item still occurs, recover from a backup

72 MenuBar list object has a malformed header.

Detailed Description: SanityCheck generates this item when it has successfully read the MenuBar list into memory but cannot understand the header of the list.

Actions: Try editing a few of the menu bars and running 4D Tools:Compact. If the item is still generated, recover from a backup.

73 Unexpected end of object while parsing the MenuBar list.

Detailed Description: SanityCheck generates this item when it encounters the end of the Menu Bar list while still expecting more information to exist.

Actions: Try editing a few of the menu bars and running 4D Tools:Compact. If the item is still generated, recover from a backup.

74 MenuBar #<1> has a malformed header.

Detailed Description: SanityCheck generates this item when it has successfully read the MenuBar into memory but cannot understand the header of the Menu Bar.

Actions: Try editing the menu bar and running 4D Tools:Compact. If the item is still generated, delete the MenuBar and recreate it.

75 MenuBar #<1> cannot be found. MenuBar list object may be the cause.

Detailed Description: SanityCheck generates this item when it cannot find the menu bar that is mentioned in the MenuBar list.

Actions: Try editing the menu bar and running 4D Tools:Compact. If the item is still generated, delete the MenuBar and recreate it.

76 Unexpected end of object while parsing a menu.

Detailed Description: SanityCheck generates this item when it encounters the end of the Menu Bar list while still expecting more information to exist.

Actions: Try editing the menus for the MenuBar and running 4D Tools:Compact. If the item is still generated, delete the Menu and recreate it.

77 Menu #<1> cannot be found.

Detailed Description: SanityCheck generates this item when it cannot find the menu that is mentioned in the MenuBar.

Actions: Try editing the menu for the MenuBar and running 4D Tools:Compact. If the item is still generated, delete the MenuBar and recreate it.

78 Menu has a malformed header.

Detailed Description: SanityCheck generates this item when it has successfully read the Menu into memory but cannot understand the header of the Menu.

Actions: Try editing the menu and running 4D Tools:Compact. If the item is still generated, delete the Menu and recreate it.

79 Unexpected end of object while parsing menu item #<1>.

Detailed Description: SanityCheck generates this item when it encounters the end of the menu item while still expecting more information.

Actions: Try editing the menu changing the menu item information. Try running 4D Tools:Compact. If the item is still generated, delete the Menu and recreate it.

80 Menu item #<1> has a zero length name.

Detailed Description: The menu item has no name. Although this is not a problem, you may wish to be aware of it, since Apple's Human Interface Guidelines advises against empty menu items.

Actions: You may choose to ignore this item. Otherwise, edit the menu and add a name for the menu item.

81 Menu item #<1> ('<2>') doesn't call a procedure.

Detailed Description: The menu doesn’t call a procedure. Although this is not a problem, you may wish to be aware of it. A menu that calls no procedure will bring the user into User Mode from Runtime Mode.

Actions: Edit the menu and add a procedure, if so desired.

82 Menu #<1> has a zero length name.

Detailed Description: The menu has no name. Although this is not a problem, you may wish to be aware of it, since Apple’s Human Interface Guidelines advises against empty menu names.

Actions: Most likely, you’ll just want to delete the menu from the menu bar.

85 Procedure '<1>' referenced by '<2>' does not exist.

Detailed Description: The menu calls a procedure that doesn’t exist in the structure file.

Actions: Either add the procedure, or delete the reference to the procedure from the menu.

86 Menu #<1>'s name is too long ('<2>').

Detailed Description: The name of the menu is too long. The maximum length is 15 chars. The name shown is truncated to 15 chars.

Actions: Edit the menu and retype the name of the menu. If the item continues to appear, delete the whole menu/menu bar and recreate it from scratch.

87 Menu item #<1>'s procedure name is too long ('<2>').

Detailed Description: The name of the procedure called by the menu is too long. The maximum length is 15 chars. The name shown is truncated to 15 chars.

Actions: Edit the menu item and retype the name of the procedure If the item continues to appear, delete the whole menu/menu bar and recreate it from scratch.

88 Menu item #<1>'s name is too long ('<2>').

Detailed Description: The name of the menu item is too long. The maximum length is 31 chars. The name shown is truncated to 31 chars.

Actions: Edit the menu item and retype the name of the menu item. If the item continues to appear, delete the whole menu/menu bar and recreate it from scratch.

89 Unrunnable code exists on line <2>: ('<1>').

Detailed Description: The procedure contains code that cannot be executed by the interpreter, or compiled by the compiler. This may occur when using 4D Mover to move procedures from one structure to another when the files mentioned in the source structures procedure don't exist in the destination structure file. It may also

occur if you simply type the partial name of a 4D file into the procedure editor. You can see the errors via a bullet (•) that appears in the procedure.

Actions: Edit the procedure and fix the offending lines of code.

90 Variable '<1>' on line <2> has gremlins.

91 Procedure '<1>' on line <2> has gremlins.

92 Local variable '<1>' on line <2> has gremlins.

93 Inter process variable '<1>' on line <2> has gremlins.

Detailed Description: A gremlin is a non-viewable ASCII character that appears in the name of an object (Variable, Procedure, etc.) in your procedure. When you accidently type a gremlin into your procedure, you can accidently change the name of an object and will never see that the name has changed.

Actions: You may choose to ignore this item. If the gremlin does exist, you should retype the whole line that the gremlin exists on. To be sure, you may wish to delete the line preceeding and line following the offending line and just retype those three lines to be sure. If the character that SanityCheck is calling a “gremlin” is a valid character in your language, use the gremlin editor (under the Settings Menu) and un-hilite the character.

94 Picture located at (<1>) appears to be damaged.

Detailed Description: The picture at the given location seems to be damaged.

Actions: If you can open the layout, you want to quickly delete the picture and quit 4D, then reopen and paste in a new copy of the picture. Remember that a picture may be damaged before you paste it in, so if after copying in a new picture this item is still generated, you may wish to re-generate the picture from scratch. If you can’t open the layout, then you should delete the whole layout and recreate it from scratch. Note: in version 3 databases, you can hold down the option key while selecting a layout from the layout list and the “thumb” is not drawn. This way, you can easily delete the layout.

95 This procedure is too complex for SanityCheck to perform a syntax check.

Detailed Description: SanityCheck has some internal limits for certain elements of syntax checking. For example, you can only nest **If** statements up to 255 times.

Actions: If you encounter this item, and your procedure looks the way you want it to, then contact Committed Software for an upgrade.

96 Syntax: Unexpected <1> on line <2>.

Detailed Description: SanityCheck generates this item for many different types of syntax errors. SanityCheck is a little stricter about certain syntax type errors than the compiler is. For example, SanityCheck will complain about the following line, whereas the compiler will not: “**C_INTEGER**($counter;)” Note that the semicolon shouldn’t really exist.

Actions: Study the line that SanityCheck mentions and keep in mind the explicit description that SanityCheck is warning you about. Fix whatever problem appears in the line.

97 Syntax: Variable '<1>' unexpected on line <2>.

Detailed Description: This item is generated when a variable is unexpected at this point in the line.

Actions: Study the line that SanityCheck mentions and pay particular attention to where the mentioned variable is. Fix whatever problem appears in the line.

98 Syntax: No right-hand-side of expression on line <1>.

Detailed Description: This item is generated when an assignment expression contains no right-hand-side. For example, the line "$myVar:=" will cause SanityCheck to generate this item.

Actions: Add the right-hand-side to the line.

99 Syntax: Expected expression on line <1>.

Detailed Description: An expression was expected but doesn't exist. For example, the following line will cause this item to be generated: "**For**". This is a syntax error because there should be some arguments to the for statement.

Actions: Study the line carefully and add the expression where necessary.

100 Syntax: Unexpected end of procedure.

Detailed Description: SanityCheck keeps track of how many **If**'s, **While**'s, etc., that you procedure has. When the procedure ends, SanityCheck verifies that they have all been balanced. If they are not, then this item is generated.

Actions: Study the procedure carefully and make sure all the **If**'s, **Case of**'s, etc., are properly balanced. Fix as necessary.

101 Syntax: Unbalanced <1> on line <2>.

Detailed Description: This item is generated when the procedure has an **If**-**End While** or **For**-**End If** type pair.

Actions: Study the procedure carefully, paying particular attention to the balancing of the item noted. Fix the line and rerun SanityCheck.

102 Syntax: Unbalanced <1> on line <2>.

Detailed Description: This item is generated when a line contains unbalanced parens, curly parentheses, brackets, etc.

Actions: Study the line carefully, paying particular attention to the balancing of the item noted. Fix the line and rerun SanityCheck.

103 Syntax: No assignment or function call on line <1>.

Detailed Description: Each line must either be empty, be a function call or contain an assignment. The given line doesn't do either of these three, and is therefore a syntax error.

Actions: Study the line carefully, and fix whatever problem appears.

104 Syntax: Left hand side of expression is a constant on line <1>.

Detailed Description: The line is an assignment line, and the left side of the assignment (or the entity being assigned) is a constant. This is illegal. For example, this occurs when the following line exists in your procedure: "3:=4".

Actions: Study the line carefully, and fix the syntax error.

105 Style: Line <2> contains unbounded expressions.

Detailed Description: This item occurs when you have unbounded expressions. This is a style issue that some programmers like to adhere to. 4D allows you to have the following line in your procedure: "**While** (i<10) & (j<10)". Although this

causes no problem for your program, some people would prefer to have the more syntactically correct: “**While** ((i<10) & (j<10))”. The simple style argument is that the boolean operators are a single argument to the **While** function.

Actions: This item only appears for people who wish to keep stricter control over some style issues. You may choose to ignore this item. To conform to this style issue, simply surround each **While**, **If**, etc. statement with bounding parenthesis.

106 Syntax: Left hand side of expression is not properly formed on line <2>.

Detailed Description: This item occurs when you have tried to use mathematical operations on the left hand side of an assignment operation. For example, the following line will cause this syntax error: “$var1+$var2:=3”.

Actions: Rewrite the line.

107 Style: '<1>' on line <2> is not explicitly declared before being used.

Detailed Description: This item occurs when you have not explicitly defined a local variable before using it. Note that this item occurs even if you define the local variable after you have first used it. You explicitly define a local variable by adding it to a compiler directive statement (**C_INTEGER**, **C_STRING**) at the beginning of your procedure. Note: Many developers want to make sure they explicitly define their local variables because certain local variables are faster than others for looping, and also they want to watch out for stack space size.

Actions: This item only appears for people who wish to keep stricter control over some style issues. You may choose to ignore this item. To conform to this style issue, simply predefine each local variable in a compiler directive statement (**C_INTEGER**, etc.) .

108 Rect at (<1>) is off screen.

109 Round rect at (<1>) is off screen.

110 Oval at (<1>) is off screen.

111 Line at (<1>) is off screen.

112 Text object at (<1>) is off screen.

113 Picture at (<1>) is off screen.

114 Variable at (<1>) is off screen.

115 Field at (<1>) is off screen.

Detailed Description: This item occurs the object appears off the screen (above, to the left or both). These objects will never be seen by the user so they might as well be deleted.

Actions: You should delete these objects, since they are wasting space. One method is to open the layout and do a select all. Then manual de-select each item (shift-click on it) that is valid. After doing this, all that is left over that is selected is the offscreen object. Pressing the delete key will delete it.

116 Layout named '<1>' is an orphan. Layout size: <2> bytes.

Detailed Description: Sometimes, a layout can be “orphaned” in your structure file. The information in the layout still exists in your structure file, yet no file claims to own the layout. Therefore, the layout is occupying space in your file, but can never be accessed. This causes no problem whatsoever, except that it eats up

disk space. The size is given so that you know how much disk space is being wasted.

Actions: There is no known way to rid yourself of these orphans, except to rebuild your structure file from scratch. It is recommended that you simply ignore this item, unless it becomes excessive.

117 Layout named '<1>' is an orphan. Layout size: <2> bytes.

Detailed Description: Sometimes, a layout can be “orphaned” in your structure file. The information in the layout still exists in your structure file, yet no file claims to own the layout. Therefore, the layout is occupying space in your file, but can never be accessed. This causes no problem whatsoever, except that it eats up disk space. The size is given so that you know how much disk space is being wasted.

Actions: There is no known way to rid yourself of these orphans, except to rebuild your structure file from scratch. It is recommended that you simply ignore this item, unless it becomes excessive.

118 '<1>' on line <2> is used before it is initialized.

Detailed Description: A local variable is being accessed before it has been initialized. A variable can be initialized by either assigning it, using it in a for loop, or defining it with a compiler directive (**C_INTEGER**, etc.).

Actions: Initialize the variable before you use it.

119 Layout size: <1> bytes.

Detailed Description: This item is generated when a layout’s total size exceeds the size set in the Tolerances dialog box.

Actions: This is an informative message only. If you have a lot of these messages, and you know why your layouts are big, then simply bump up the tolerance in the Tolerances dialog.

120 Picture size: <3> bytes. (loc: <1>).

Detailed Description: This item is generated when a picture’s total size exceeds the size set in the Tolerances dialog box.

Actions: This is an informative message only. If you have a lot of these messages, and you know why your pictures are big, then simply bump up the tolerance in the Tolerances dialog to skip over these.

121 Layout object '<2>' is too small (siz: <3>, loc: (<4>)).

Detailed Description: This item is generated when the total size of a layout object (in square pixels) is less than or equal to the size set in the tolerances dialog box.

Actions: This is an informative message only. If you have a lot of these messages, and you know why you have lots of tiny objects, then simply bump up the tolerance in the Tolerances dialog to skip over these.

123 Local variable '<1>' is explicitly declared but never used.

Detailed Description: This item is generated when you explicitly declare a local variable, but never use it. A local variable is explicitly declared when it appears in a compiler directive statement (**C_INTEGER**, etc.). See “Common Questions” on page 44 for a discussion regarding why this item is very useful for optimizing a database.

Actions: This is an informative message only. If you have a lot of these messages, and you don't care about having defined variables that you don't use, then simply ignore the message, or use the Reported Items dialog to make this item unreported.

124 Layout variable '<2>'<1> appears <3> times.

Detailed Description: A layout variable is used several times in the same layout. This is not a problem, but may be something that you wish to be aware of.

Actions: This is an informative message only. If you have a lot of these messages, and you don't care about having the same variable appear several times on the same layout, then simply ignore the message, or use the Reported Items dialog to make this item unreported.

125 '<1>' on line <2> is used in an external call before it is initialized.

Detailed Description: This is similar to item 118, however SanityCheck cannot tell whether the external is setting the value of the local variable. So to allow you to quickly tell the difference between local variables that are not initialized and those that *may* not be initialized, this item is generated.

Actions: This message may be ignored if you know that the external is setting the value of the local variable in question. Many developers assign values to local variables before passing them off to an external to avoid this item.

126 Subfile with first field named '<1>' has no parent file.

Detailed Description: When SanityCheck encounters a subfile in the data dictionary, it checks to make sure that the parent exists. If the parent doesn't exist, then SanityCheck generates this item. This item seems to appear on old version 1 databases that were converted to version 2 databases. It appears that some of the information about the subfile-parentfile relationship was not properly saved into the version 2 database. SanityCheck gives you the name of the first field in the file who has lost its parent, because that is all the information it has to work with.

Actions: Do not try to run 4D Tools first (it has had problems with this scenario). Instead, open the structure file in design mode and change the name of the file. If you're not sure exactly which one is having problems, just change them all (you can change them back later). Changing the file name simply forces a re-write of the data dictionary to disk. After you change the names, quit 4D and then run the structure file through 4D Tools. Then check with SanityCheck to verify that all is together. If this doesn't work, try changing all the field names and then run through 4D Tools:Compact. If that doesn't work, recover from a backup.

127 <1> Layout Var '<2>' has same key-equivalent as '<3>'.

Detailed Description: SanityCheck generates this item when two objects on the same page of the same layout have the same key-equivalent.

Actions: This is not necessarily a problem, yet you may wish to be aware of these situations and handle them accordingly.

128 Unknown token (8)

Detailed Description: This item occurs when an unknown token of type 8 is encountered by SanityCheck. It appears that under certain situations, 4D Insider:Global Search and Replace appends a token to the end of every line in the procedure being modified. This token is of type 8. If you open the procedure, press command-enter, then save the procedure, all of the appended tokens are

removed. This token has never been seen by Committed Software in any other situation except the one just described. Since the procedure editor appears to ignore and eventually delete these tokens, SanityCheck reports them to you. Neither Committed Software, nor any other party has been able to confirm that these extra tokens cause any problem.

Actions: This is not necessarily a problem, yet you may wish to be aware of these situations and handle them accordingly. If you open the procedure, press command-enter, then save the procedure, the extra tokens go away. If they do not go away for that procedure, then try copying the procedure to a text editor, deleting the old procedure, creating a new one, and pasting the text back in. If the item still appears for the procedure, delete the procedure and retype it in by hand. You may use the Reported Items dialog box to make this item Unreported, if you wish to ignore them.

129 Layout has no pages

Detailed Description: This item occurs when a layout is found to have no pages. This is usually indicative of a severely damaged layout.

Actions: You need to delete the layout, and recreate it from scratch.

130 Number of Lines

Detailed Description: The number of lines in the script or procedures exceeds the tolerance. There is no problem with large procedures, this is simply here to help you manage the size of your structure files.

Actions: You don’t need to do anything. However, if you would like to change the default tolerance, you may do so by selecting the “Tolerances” menu item from the “File” menu.

131 Num Objects

Detailed Description: The number of objects in the layout exceeds the tolerance. Having a large number of objects may slow down layout display time.

Actions: You don’t need to do anything. However, if you would like to change the default tolerance, you may do so by selecting the “Tolerances” menu item from the “File” menu.

132 No Lines

Detailed Description: The procedure or script has no lines.

Actions: You may wish to delete the script or procedure since it doesn’t do anything.

133 No Objects

Detailed Description: The layout has no objects in it. Sometimes, a blank layout is used by a developer in a **PRINT LAYOUT** call, to add pixels to the middle of a hand-built report. A blank layout is also sometimes used as an input layout in Main-Event-Loop type programming (the layout proc might just have a cancel in it).

Actions: You may wish to delete this layout. However, first check to make sure that it isn’t referenced anywhere in your database (perhaps in a **PRINT LAYOUT** call, or an **INPUT LAYOUT**). You can use the SanityCheck Find feature to look for the layout in your structure file.

134 Layouts

Detailed Description: STATISTICS: Gives the total number of layouts in your structure file. Includes all orphaned and unreferenced layouts.

Actions: Informational Only.

135 Procs/Scripts

Detailed Description: STATISTICS: Gives the total number of procedures and scripts in your structure file. Includes all orphaned and unreferenced procedures and scripts.

Actions: Informational Only.

136 Total lines (includes blank & comment-only lines)

Detailed Description: STATISTICS: Gives the total number of lines of code in your structure file. It includes the blank and comment-only lines.

Actions: Informational Only.

137 blank or comment-only lines

Detailed Description: STATISTICS: Gives the total number of blank or comment-only lines in your structure file. This is useful to know how much “dead” space is in the “number of lines of code” statistic above. Some folk don't want to count blank lines as “code”.

Actions: Informational Only.

138 Files

Detailed Description: STATISTICS: Gives the total number of files in your structure file.

Actions: Informational Only.

139 Menu Bars

Detailed Description: STATISTICS: Gives the total number of menu bars in your structure file.

Actions: Informational Only.

140 Balloon Help Items

Detailed Description: STATISTICS: Gives the total number of balloon help items in your structure file.

Actions: Informational Only.

141 seconds to process

Detailed Description: STATISTICS: Gives the total number seconds SanityCheck took to process your structure file. This is an approximate number, and will tend to err a tad low (of course, making SanityCheck seem all that much faster).

Actions: Informational Only.

142 CPU seconds to process

Detailed Description: STATISTICS: Gives the total number of CPU seconds SanityCheck took to process your structure file. This differs from the total number of seconds to give you an idea how much time SanityCheck is allowing the operating system to do other things. SanityCheck tries to be a ‘good’ Macintosh applica-

tion by giving time to other processes. But due this goodness, SanityCheck runs a little slower. If you are doing comparisons between runs of SanityCheck, use this item as your base number. If you just want to know how long it took SanityCheck, item 141 is closer to your sit-and-wait time.

Actions: Informational Only.

143 '<1>' on line <2> contains the reserved character '<3>'

Detailed Description: A reserved character is a “$” or a “Þ”. Even though a variable can have these characters embedded in the variable name, it can be confusing to read.

Actions: You may wish to take these characters out of the variable name.

144 Line <1> calls "Self".

Detailed Description: “**Self**” should only be used in a script, not in a procedure or layout procedure. Calling self in a global procedure or layout procedure will yield undefined results.

Actions: Move the reference to Self into a script or restructure the procedure to not make the call to Self.

145 Math on line <1> has a suspicious operator order

Detailed Description: The order of execution is not explicitly defined. For example, 4D resolves “4+3*2” as “14” instead of the more common “24”. Your expression has a (+, -) sign before a (*, /, %, etc.) sign. NOTE: Item 146 is generated when a math expression is not parenthesized, but the math reads correctly regardless of 4D’s lack of operator precedence processing. Item 145 is generated when there is potential confusion due to 4D’s lack of operator precedence processing. For example the expression “4+3*2” would cause item 145 to be generated because 4D will generate 7 instead of the more common reading of “24”. The expression “4*3+2” would generate item 146 because both the standard operator precedence evaluation and 4D’s evaluation of the expression yield the same result. In short, pay more attention to item 145, and use item 146 as a style issue.

Actions: Verify that the math is correct and parenthesize the expression for clarity.

146 Style: Math on line <1> is not explicitly parenthesized

Detailed Description: The order of execution is not explicitly defined. For example, 4D resolves “4+3*2” as “14” instead of the more common “24”. Many developers wish to explicitly define which they mean by parenthesizing, as in “(4+3)*2” or “4+(3*2)”. NOTE: Item 146 is generated when a math expression is not parenthesized, but the math reads correctly regardless of 4D’s lack of operator precedence processing. Item 145 is generated when there is potential confusion due to 4D’s lack of operator precedence processing. For example the expression “4+3*2” would cause item 145 to be generated because 4D will generate 7 instead of the more common reading of “24”. The expression “4*3+2” would generate item 146 because both the standard operator precedence evaluation and 4D’s evaluation of the expression yield the same result. In short, pay more attention to item 145, and use item 146 as a style issue.

Actions: Verify that the math is correct and parenthesize the expression for clarity.

147 Object '<1>' is named '<2>'

Detailed Description: Sometimes, SanityCheck cannot find the name of an

object when generating an item (most often, this happens during the very beginning of the scan — during object map verification). When SanityCheck can't find the name of an object, it will display the internal ID for the object. In some cases, SanityCheck can later figure out the name of the object as more information becomes available. When this happens, SanityCheck generates item 147 to further clarify the earlier item. So whenever you get an item that refers to an object named “CC4D:15532” or something similar, look through the log for an item 147, as SanityCheck may have later found out what the name of “CC4D:15532” really is.

Actions: None.

150 Layout '<1>' is not explicitly referenced.

Detailed Description: The named layout is never explicitly referenced in your code. An explicit reference occurs in two ways: (1) using a 4D command (2) using included layouts. Here is the list of 4D commands that will cause a layout to be explicitly referenced: **INPUT LAYOUT**, **OUTPUT LAYOUT**, **DIALOG**, **PRINT LAYOUT**, **PAGE SETUP**, **ADD SUBRECORD**, **MODIFY SUBRECORD**, **SEARCH BY LAYOUT**. For a layout to be explicitly referenced, not only do you need to use the above commands, but you need to explicitly define the file and the full layout name in the command. For example “**DIALOG**([People];"Output")” is an explicit reference to the layout “Output” in file [People]. However, “**DIALOG**(gFilePtr»;"Output")” is not an explicit reference, because SanityCheck doesn’t know what value is in gFilePtr. The following is also not an explicit reference, because SanityCheck cannot know what is in the gLayName variable: “**DIALOG**([People];gLayName)”.

Actions: Most likely nothing. Many developers build their layout names on the fly, which makes this feature of SanityCheck almost useless. However, for those developers that always explicitly refer to their layouts, this feature can be quite helpful in removing dead layouts from their structure file. You should be very careful that the layout isn’t referred to in some way before deleting it. Even if you don’t explicitly refer to your layouts, and assuming your know your structure file well, just looking through the list of unreferenced layouts can help you find layouts that are unused. You can turn off this feature if it is annoying to you.

151 File <1> has <3> layouts with the same name ('<2>').

Detailed Description: You have more than one layout with the same name in the same file in your structure file.

Actions: You should delete or change the name of the layouts until all are unique.

152 File <1> has <3> fields with the same name ('<2>').

Detailed Description: You have more than one field with the same name in the same file in your structure file.

Actions: You should change the name of the fields until all are unique. Since you cannot delete field names, sometimes developers change the name of all unused fields to something like “unused”. If this is the case for you, just ignore this item — it will cause no problem if the fields are never used in your code.

153 File <1> has a field which is a reserved word ('<2>')

Detailed Description: A field can have the same name as a 4D function, but it could cause a conflict in a layout procedure or layout script. A field name of “length” might get confused with the 4D function **Length**.

Actions: Simply change the field name to avoid potential conflicts.

154 File <1> has a field which is the same name as a procedure ('<2>')

Detailed Description: A field can have the same name as a procedure, but it could cause a conflict in a layout procedure or layout script. A field name of "TheLength" might get confused with a procedure "TheLength".

Actions: Simply change the field or procedure name to avoid potential conflicts.

155 <2> procedures have the same name ('<1>').

Detailed Description: Some procedures have the same name, which can cause unpredictable results.

Actions: Simply change the procedure names to avoid potential conflicts.

156 Script refers to layout '<1>' for file '<2>' which doesn't appear to exist.

Detailed Description: When you explicitly refer to a layout using a 4D command, SanityCheck verifies that the layout exists. If it doesn't exist, this item is generated. Here is the list of 4D commands that can cause this item to be generated: **INPUT LAYOUT**, **OUTPUT LAYOUT**, **DIALOG**, **PRINT LAYOUT**, **PAGE SETUP**, **ADD SUBRECORD**, **MODIFY SUBRECORD**, **SEARCH BY LAYOUT**.

Actions: Either create the layout, or remove the reference to the layout.

157 Layout refers to included layout '<1>' for file '<2>' which doesn't exist.

Detailed Description: The included layout doesn't exist.

Actions: Either create the layout, or change the included layout reference in the parent layout. You can do the latter by opening the parent layout, double clicking on the included layout area, and re-selecting the appropriate included layout(s).

158 Splash screen picture appears damaged

Detailed Description: This item occurs when the splash screen associated with a given menu has been damaged. This can cause random crashing within your database, even if you don't use this menu splash screen (it can be read in and buffered even if it's not used).

Actions: You should at least clear this picture from the splash screen. Open the menubar, choose "Show Custom Menus" from the "Menu" menu, then choose "Clear" from the Edit menu. Immediately quit 4D and relaunch.

159 Menubar #<1> doesn't exist in database.

Detailed Description: The numbered menu bar is referenced, yet has been deleted or never existed.

Actions: Open layout, select "Menu Bar" from "Layout" menu and set the menu bar. Note: see tolerance settings to ignore menu numbers over a certain value.

160 Resource file not found.

Detailed Description: The resource file is not present.

Actions: This has been deactivated for the initial release of 2.0. It annoyingly complains when the Proc.ext file is missing from the structure's directory. Since SanityCheck will complain if a referenced external doesn't exist, this is somewhat redundant.

161 Resource fork is empty.

Detailed Description: The named resource fork cannot be opened (i.e., is empty). This can be a very serious problem — 4D will crash if the structure file has

no resource fork (even if it's empty, it has to be present).

Actions: Verify with ResEdit that the resource fork is empty (it should warn you as you try to open the file), then create it (answer "yes" when ResEdit asks you to create the resource fork).

162 Resource Map out of bounds

Detailed Description: Serious resource fork damage. The header of the resource fork claims that the resource map exists outside of the bounds of the file itself.

Actions: Resource fork is severely damaged, use ResEdit to try to salvage. If you cannot salvage any part of the resource fork, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

163 Resource Data out of bounds

Detailed Description: Serious resource fork damage. The header of the resource fork claims that the resource data exists outside of the bounds of the file itself.

Actions: Resource fork is severely damaged, use ResEdit to try to salvage. If you cannot salvage any part of the resource fork, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

164 Resource type list out of bounds

Detailed Description: Serious resource fork damage. The header of the resource fork claims that the list of resource types exists outside of the bounds of the file itself.

Actions: Resource fork is severely damaged, use ResEdit to try to salvage. If you cannot salvage any part of the resource fork, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

165 Resource name list out of bounds

Detailed Description: Serious resource fork damage. The header of the resource fork claims that the list of resource names exists outside of the bounds of the file itself.

Actions: Resource fork is severely damaged, use ResEdit to try to salvage. If you cannot salvage any part of the resource fork, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

166 Resource '<1>' reference list bad (idx: <2>)

Detailed Description: Serious resource fork damage. Each resource type has a list of resources that have that type (the "reference list"). The resource map claims that the reference list for this type exists outside of the bounds of the file.

Actions: Resource fork is severely damaged, use ResEdit to try to salvage. The "idx" mentioned is the order that this type appears in the file. If the name doesn't make sense, try using "View:By Order In File" in ResEdit to figure out which is damaged. If you cannot salvage any part of the resource fork, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

167 Resource ‘<1>’ has a damaged name.

Detailed Description: The reference list (see description from 166) also contains information regarding the place to find name of each resource of a given type. The given resource references a name that exists outside of the file.

Actions: Resource fork is damaged. This may be an indication of serious damage that hasn’t really come to fruition, so you need to attend to this item immediately. Try using ResEdit to rename the resource. Try deleting the resource altogether. Try deleting all resource and re-installing your externals. If that fails, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

168 Resource ‘<1>’ has data out of bounds.

Detailed Description: The reference list (see description from 166) also contains information regarding the place to find data of each resource of a given type. The given resource references data that exists outside of the file.

Actions: Resource fork is damaged. This may be an indication of serious damage that hasn’t really come to fruition, so you need to attend to this item immediately. Try deleting and recreating the resource. Try deleting all resource and re-installing your externals. If that fails, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

169 Resource ‘<1>’ and ‘<2>’ overlap.

Detailed Description: The two resources claim to occupy the same space on disk.

Actions: Resource fork is severely damaged. Be aware that ResEdit may not catch this particular damage. Try deleting and recreating the resources. Try deleting all resource and re-installing your externals. If that fails, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See the About Box of DataForkCopy for instructions.

170 Resource ‘<1>’ extends beyond the end of file.

Detailed Description: The resource claims to have more data than exists in the file itself.

Actions: Resource fork is damaged. Try deleting and recreating the resource. Try deleting all resource and re-installing your externals. If that fails, use DataForkCopy (included in this release of SanityCheck for free) to copy just the data fork of your structure file. See about box of DataForkCopy for instructions.

171 Resource ‘<1>’ appears to be damaged.

Detailed Description: The resource has suffered internal damage, but the rest of the resource fork may be OK. This only appears for PICT resources in version 2.0 of SanityCheck (perhaps others in the future).

Actions: Resource is damaged. Try deleting and recreating the resource. Note that if this is a PICT resource, ResEdit and Resourcer will not pick up the damage (and may crash). So do your work quickly.

172 Resource ‘<1>’ owned by ‘<2>’ doesn’t appear to exist.

Detailed Description: SanityCheck is verifying the externals and packages installed in your structure or Proc.ext files. It is complaining that one of the exter-

nals or packages appears to require a resource that doesn't exist.

Actions: Try re-installing the external or package. If that doesn't help, try contacting the external developer to verify that the external is in working order.

173 Resource '<1>' owned by '<2>' is referenced more than once.

Detailed Description: SanityCheck is verifying the externals and packages installed in your structure or Proc.ext files. It is complaining that one of the externals or packages references a resource that another external or package also references.

Actions: Try re-installing the external or package. If that doesn't help, try contacting the external developer to verify that the external is in working order.

174 External '<1>' appears more than once.

Detailed Description: The named external appears more than once in your resources. This can cause problems since it is undetermined which will be executed by 4D at runtime.

Actions: If you've accidentally installed the same external or package in two places, just delete the extra copy. If you have a conflict, choose which you will use in your structure and delete the other.

175 External '<1>' from '<2>' appears more than once.

Detailed Description: The named external from the named package appears more than once in your resources. This can cause problems since it is undetermined which will be executed by 4D at runtime.

Actions: If you've accidentally installed the same external or package in two places, just delete the extra copy. If you have a conflict, choose which you will use in your structure and delete the other. Unfortunately, you may need to contact the package developer to rename the externals to be more unique, since names like "GetPict" might be in many "Tool" type packages.

176 External '<1>' claims to own <2> resources, but only <3> exist.

Detailed Description: The named external claims to own the amount of resources shown, but only the given amount actually do exist. This may be an old-style external whose bundle (4BNX, 4BND, etc.) resource is of an old style (very outdated).

Actions: Try reinstalling the external using the original installer. If that doesn't help, try contacting the external developer to see if there is a more recent version of the external.

177 External Package '<1>' has no related bundle resource.

Detailed Description: Each external package must have a related '4BNX' resource which describes all of the resources owned by the package. The named external package has no bundle resource.

Actions: Try reinstalling the package using the original installer. If that doesn't help, try contacting the package developer to see what the problem might be.

178 Bundle Resource '<1>'('<2>') has no related package resource ('4DPX').

Detailed Description: Each 4BNX resource (bundle resource for packages) should have an accompanying 4DPX resource (package code resource). The package resource is missing.

Actions: Try reinstalling the external using the original installer. If that doesn't

help, try contacting the external developer to see what the problem might be.

179 Resource ‘<1>’ is not owned by an external.

Detailed Description: After scanning all resources and figuring out which are “owned” by which package and external, these are the resources left over. Be aware that some external developers do not properly “tag” each resource that they require, so don’t go deleting resources willy-nilly. Also, SanityCheck does not track resources used by the international MENU system in 4D (yet). Plus if you installed custom sounds and pictures, etc., then SanityCheck will of course note those as not being owned.

Actions: There’s no need to do anything — this is purely informational. You might want to peruse this list and see if anything looks out of the ordinary. If you’re not really a power user of 4D or ResEdit, you may just want to turn this item off, since it can be somewhat annoying, complaining about this-and-that for no reason.

180 Resource ‘<1>’ has no related external package.

Detailed Description: Each ‘Soso’ resource must have a related ‘4DPX’ (package) resource. The ‘Soso’ resource is used to allow special external packages to run on the server machine.

Actions: Try reinstalling the external using the original installer. If that doesn’t help, try contacting the external developer to see what the problem might be.

181 Bundle Resource ‘<1>’(‘<2>’) has no related package resource (‘4DEX’).

Detailed Description: Each 4BND resource (bundle resource for packages) should have an accompanying 4DEX resource (external code resource). The external resource is missing.

Actions: Try reinstalling the external using the original installer. If that doesn’t help, try contacting the external developer to see what the problem might be.

182 Procedure has the same name as an external

Detailed Description: The named procedure has the same name as an installed external, which may cause problems discerning which is to be executed at runtime.

Actions: Rename your procedure to be unique.

183 Procedure calls external ‘<1>’ that doesn’t appear to exist (line: <2>)

Detailed Description: The procedure is calling an external that is not in either the Proc.ext or the structure file.

Actions: Make sure the structure file and Proc.ext file are in the same directory (if you are using a Proc.ext file). Be aware that SanityCheck does not check for externals installed in the 4D application (or runtime).

184 Procedure has no code.

Detailed Description: The procedure either has no code — just blank lines and comments.

Actions: Comments can be quite valuable, so be sure you know what you’re deleting before you delete it...

185 Picture has a depth greater than 8 bits per pixel (depth: <1>).

Detailed Description: The given picture has a depth that cannot be displayed by most displays.

Actions: This is for your information only. You may wish to use a painting program (PhotoShop, etc.), to reduce the pixel depth of the picture. Be aware that printers can print pictures that have greater depths.

186 Field '<1>' overlaps another field.

Detailed Description: The field overlaps another field on the screen. For memory constraints and speed, SanityCheck does not track the second overlapping field's name.

Actions: You may have some reason to make the fields overlap, but in most cases, overlapping fields are something that you want to take care of.

187 Layout Field '<2>'<1> appears <3> times.

Detailed Description: The field appears more than once on the layout.

Actions: This is for your information only — there are many instances where this is desirable to do.

188 File procedure exists in file '<1>'.

Detailed Description: This is simply a warning that the file procedure exists.

Actions: Many developers forget that file procedures exist. Sometimes when you're debugging, you may forget that you have a file procedure that is affecting the layout. This feature is to try to remind you of these oft-forgotten procedures. There is no known way to delete a file procedure, so you'll probably see this in conjunction with item 184 quite often.

189 '<1>' should be indexed (it's the ONE field of an automatic Many->One).

Detailed Description: 4D normally automatically indexes in this situation, and should have indexed this field.

Actions: Open the structure file, open this file/field and select "indexed". Redrawing the relation line might also fix this.

190 '<1>' may need to be indexed (it's the MANY field of a One->Many).

Detailed Description: The named field is part of a relation, but isn't indexed.

Actions: If you call **RELATE MANY**, or if you often search this field, you may wish to index it.

191 '<1>' should be unique (it's the ONE field of an automatic Many->One).

Detailed Description: The named field is the "one" part of a Many ? One. In most cases, this field should be unique, to help ensure against bad data being entered into 4D.

Actions: It is not necessary to do this, but strongly suggested. It helps preserve relational integrity.

192 '<1>' may need to be unique (it's the ONE field of a Many->One).

Detailed Description: The named field is the "one" part of a Many ? One. In most cases, this field should be unique, to help ensure against bad data being entered into 4D. This is less strict than item 191 because it is not part of an automatic relationship.

Actions: It is not necessary to do this, but suggested. It helps preserve relational integrity.

193 '<1>' may need to be indexed (it's the ONE field of a Many->One).

Detailed Description: The named field is the "one" part of a Many ? One. You may want to index this field.

Actions: If you call RELATE ONE, or if you often search this field, you may wish to index it.

194 '<1>' should be indexed (it's the MANY field of an automatic One->Many).

Detailed Description: 4D normally automatically indexes in this situation, and should have indexed this field.

Actions: Open the structure file, open this file/field and select "indexed". Redrawing the relation line might also fix this.

195 '<1>' should not be unique (it's the MANY field of a One->Many).

Detailed Description: Under most circumstances, you should not have a unique field as the MANY part of a One ? Many relationship.

Actions: Verify that you don't want the field to be unique, then open the field and uncheck the unique checkbox.

196 '<1>' (<2>) is not the same type/size as '<3>' (<4>).

Detailed Description: The type and size of the given fields are no the same, yet they are part of a relationship.

Actions: There are situations where you may want your relations to have different types or sizes, yet it is undesirable, since you may have a situation where your relations do not work properly. To be sure that your relations work properly, make all related fields have the same type and size.

197 '<1>' refers to nonexistent file as part of a relation (file idx: <2>).

Detailed Description: The relationship is damaged, and you need to fix it. The file num shown is the root file for the destination of the relationship (according to the relations info). If the relationship is to a subfile, then the root file is the parent of the subfile that is not itself a subfile.

Actions: (1) delete the relationship. (2) Quit4D (3) Re-create the relationship (4) Quit4D (5) Re-run SanityCheck.

198 '<1>' refers to nonexistent field number <3> in file <2> as part of a relation.

Detailed Description: The relationship is damaged, and you need to fix it.

Actions: (1) delete the relationship. (2) Quit4D (3) Re-create the relationship (4) Quit4D (5) Re-run SanityCheck.

199 Cannot read header of style list.

Detailed Description: The style list is damaged.

Actions: If you can open the database, try deleting all styles and re-entering them. Try 4D Tools:Compact.

200 Unexpected end of object encountered in Style list.

Detailed Description: The style list is damaged.

Actions: If you can open the database, try deleting all styles and re-entering them. Try 4D Tools:Compact.

201 Style has no name.

Detailed Description: The style has no name.

Actions: Open the style editor (preferences in design mode) and name the style.

202 Style ‘<1>’ doesn’t appear to be used.

Detailed Description: The style is not referenced by **SET FILTER**, **SET FORMAT** or in a layout.

Actions: Note that the command **SET FILTER** may pass a variable as the filter name which SanityCheck cannot check. Be careful when deleting styles that you think may not exist.

203 Style ‘<1>’ has no format.

Detailed Description: The style is empty.

Actions: It’s OK to have no format: this is just for your information.

204 ‘<1>’: Style ‘<2>’ not in database.

Detailed Description: The referenced style is not in the database.

Actions: SanityCheck has an internal limit of 127 characters for the name of a style. So assuming your style name is less than 127 characters, then you have a missing style. Either add the style to the style list, or change the style for this object.

205 ‘<1>’: Style ‘<2>’ not in database.

Detailed Description: The referenced style is not in the database.

Actions: SanityCheck has an internal limit of 127 characters for the name of a style. So assuming your style name is less than 127 characters, then you have a missing style. Either add the style to the style list, or change the style for this command. Be aware that if this command is “building” the style name (**SET FORMAT**("|"+$MyVar)), SanityCheck will be confused.

206 Key for item <1> is not in standard alphabet (key: '<2>').

Detailed Description: The given key is not in the standard set of keys normally used for a menu. This can occur either because you are using an odd-key combination (which is OK), or because there has been some damage to this menu's key command.

Actions: If this is the Key you want here, do nothing. If you realize that this is not as you intended, then simply change the command key equivalent.

207 Duplicate Table/File number <1> found. 4D always uses the last one, which renders '<2>' an orphan.

Detailed Description: The given table/file number was found already claimed by a previous table. When this occurs, 4D always uses the second (or last if there are several) table/file. This renders the named object as an orphan.

Actions: You can leave this orphan in, as it should not affect the runtime of your program at all. You may also remove it by using SanityCheck's Browser feature. Simply note the object ID, go to the browser, select the object, and press delete. Always make a backup before performing any modification to your structure file. You may also decide to run 4D Tools:Compact after this operation to clean up the bitmap.

208 Resource bundle '<1>' is too big for SanityCheck to manage.

Detailed Description: SanityCheck has an internal limit of 1024 entries in the bundle resources. Most likely, this reflects damage in the bundle resource.

Actions: Try re-installing the plug-in / external. If this item still occurs, contact the external vendor to determine if this is a problem with their external or if Sanity-Check needs to be revved to handle larger bundle sizes. Contact Committed Software if the latter is true.

209 Resource '<1>' has duplicate ID.

Detailed Description: There are two resources with the same resource ID.

Actions: Use Mac resource editing tool to fix this problem.

210 4DX Folder ('<1>') contains a Folder ('<2>').

Detailed Description: This item is only reported on the Macintosh, and is merely intended to help you clean up your Mac4DX Folder.

Actions: Nothing, this is for your information only.

211 4DX File '<1>' is of wrong type ('<2>').

Detailed Description: Mac Version Only: The 4DX file is verified to be of the proper external type. On the PC, this is impossible to do.

Actions: Try re-installing the external. If you still get this item, ask the plugin vendor if this is normal behaviour. If it is, perhaps turn off this item.

212 '§' reference used on line <1>.

Detailed Description: This token is not encouraged by ACI for variable dereferencing.

Actions: Rewrite the line to remove this token.

213 <1> object has an off-screen window.

Detailed Description: The given object's window is off screen.

Actions: You may fix this from the Browser - open the object and change the rectangle to be on-screen.

214 The general info area (PR4D) has a bad tag header (it isn't 'PR4D').

Detailed Description: The header for the general information area is incorrect.

Actions: This may indicate serious damage in your structure file. You may most likely need to recover from backup.

215 User name is too long (user: <1>, len: <2>, maxlen: <3>)

Detailed Description: The user name claims to be larger than is possible. This most likely indicates damage in the password area of your structure file, and should be addressed immediately.

Actions: Try re-entering the user name in the password editor. If this fails, recover from backup.

216 Method/Proc name is too long (user: <1>, len: <2>, maxlen: <3>)

Detailed Description: The method/proc name that is to be used as a startup procedure is too long, and indicates damage in your password area.

Actions: Try re-entering the method/procedure name. If this fails, recover from

backup.

217 Rect for user window is invalid. (user: <1>, rect: <2>)

Detailed Description: The rectangle for the user window (ie, user-mode window) is invalid.

Actions: Log into the database as this user and make sure the user mode window is visible.

218 Last day user accessed is incorrect (user: <1>, day: <2>, max: 31).

Detailed Description: The last access day is incorrect (should be less than or equal to 31).

Actions: Log into the database as this user.

219 Last month user accessed is incorrect (user: <1>, month: <2>, max: 12)

Detailed Description: The last access month is incorrect (should be less than or equal to 12).

Actions: Log into the database as this user.

220 Default owner group for objects created by this user is invalid (user: <1>, value: <2>)

Detailed Description: When the user creates an object, this field tells 4D what group to assign that object to. The group doesn't exist.

Actions: Open the password editor, edit this user. select a group from the popup of groups. If you cannot edit the user, try creating groups until you can edit the user, then change the popup.

221 Linked counters are different. (group: <1>, link1: <2>, link2: <3>)

Detailed Description: In the password structure, there are two fields that contain the same information. The 'link' information means, "which other objects (groups, users) are linked to this object". The link counter, therefore, tells how many objects are linked to this object.

Actions: This may be OK - SanityCheck developers have only seen the two linked counters being the same, and other situations should be suspect. Contact Committed Software for info if you see this.

222 Group name is too long (group: <1>, len: <2>, maxlen: <3>)

Detailed Description: The group name is too long.

Actions: Try re-entering the group name in the password editor.

223 Group owner user index is invalid (group: <1>, user: <2>, max: <3>)

Detailed Description: Each group must have an owner. This owner is defined internally as an index to the user that owns the group. SanityCheck has found this number to be invalid as it is greater than the number of users.

Actions: Edit this group and change the owner.

224 Unexpected end of record while parsing group #<1> (at link <2>)

Detailed Description: SanityCheck ran out of data while processing the given group. The 'at link' refers to which related item was being processed at the time.

Actions: You should try to edit the groups, changing the names of the groups and saving (which re-writes the lists). If this fails, recover from backup.

225 Group link index is invalid (group: <1>, link: <2>, max: <3>)

Detailed Description: Groups can contain other groups. This is accomplished by keeping a list of indexes of the groups contained. SanityCheck has found that this group index is invalid as it is greater than the number of groups.

Actions: Edit the group and verify that it looks OK. Try changing the name of the group to force a re-write of the group. If that fails, recover from backup.

226 Cannot load group #<1>

Detailed Description: The group is unloadable. Most likely, this is due to a bad group index encountered by SanityCheck.

Actions: Try adding a new group. Try editing existing groups and changing names to force a re-wriet of the password area. If this fails, recover from backup.

227 Password header static data is incorrect (is: <1>, should be 41120)

Detailed Description: There is a static number at the beginning of the password area that SanityCheck developers have always seens as 41120 (0xA0A0). If this number doesn't exist, then your password table is highly suspect.

Actions: Check with Committed Software to see if you require a new version of SanityCheck.

228 Rect for Table/File window is invalid. (user: <1>, rect: <2>)

Detailed Description: The rectangle for the small Table or File window you find in user mode is incorrect.

Actions: Log in as this user and make sure you can see this window.

229 The 4D Prefs (PR4D) object is missing from the structure file.

Detailed Description: All 4D structure files should have a PR4D object. Your structure is severely damaged if SanityCheck cannot find it.

Actions: Try recover-by-tags / compact. If that fails, recover from backup.

230 The Password object is missing from the structure file.

Detailed Description: All 4D structure files should have a password object. Your structure is severely damaged if SanityCheck cannot find it.

Actions: Try recover-by-tags / compact. If that fails, recover from backup.

231 The Table/File objects are missing from the structure file.

Detailed Description: All 4D structure files should have Table/File information object. Your structure is severely damaged if SanityCheck cannot find it.

Actions: Try recover-by-tags / compact. If that fails, recover from backup.

232 The Password list is damaged.

Detailed Description: The secondary password table is severely damaged. SanityCheck can fix this by deleting this table (and 4D will recreate it for you).

Actions: Make a backup. Allow SanityCheck to fix, open with 4D to complete fix.

233 Fixed.

Detailed Description: This item is generated when your structure file has been fixed.

Actions: No actions necesary. This is for your information only.

234 Form/Layout counter is incorrect. (Table/File: <1> count: <2>)

Detailed Description: There is a counter that defines how many Forms/Layouts exist for the given Table/File. This item is generated when that counter appears incorrect.

Actions: Try adding a new form to this table. Try deleting forms. If you still have this error, recover from backup.

235 Unexpected end of record while byte swapping. <1>

Detailed Description: This is a general error generated by the low level byte swapping code. This error should be followed by another item, giving you more information about the item and how to address it.

Actions: See items that occur after this oe to determine which object is damaged.

236 Cannot load Method/Procedure number #<1>.

Detailed Description: There are less Methods/Procedures in the structure file than are defined in the PR4D resource.

Actions: Try adding a new method/procedure to your structure file. If this doesn't fix your problem, recover from backup.

237 There is no list of Methods/Procedures in this structure.

Detailed Description: All 4D structure files should have a list of methods object.

Actions: Try opening the structure and adding a method. If this doesn't work, recover by tags.

238 There are <2> lists of Methods/Procedures - there should only be 1 list.

Detailed Description: SanityCheck has found more than 1 list of methods/procedures. There should only be one list, and the others are ignored by 4D.

Actions: If you wish, you can remove the extra lists in the Browser.

239 The constant type #<1> doesn't exist. (line <2>)

Detailed Description: The constant type (like 'TCP Port') doesn't exist. This could be due to you removing a plug-in that defined that constant.

Actions: The constant types are stored in '4DK#' resources in your structure file, in 4D's executable, and in your plug-ins. You need to determine where this constant type was defined. The most common problem occurs when you stop using a plug-in, yet your structure file still uses some constants that the plug-in defined. If you cannot find the constant, you will need to re-type the line of code to fix it.

240 The constant '4DK#:<2>' #<1> doesn't exist. (line <3>)

Detailed Description: The constant type (like 'TCP Port') doesn't exist. This could be due to you removing a plug-in that defined that constant. This is similar to 239, but differs in that only one item appears to be missing from the list, whereas in [239], the whole type is missing.

Actions: The constant types are stored in '4DK#' resources in your structure file, in 4D's executable, and in your plug-ins. You need to determine where this constant type was defined. The most common problem occurs when you stop using a plug-in, yet your structure file still uses some constants that the plug-in defined. If you cannot find the constant, you will need to re-type the line of code to fix it.

241 SCAN ABORTED DUE TO ERROR CONDITION

Detailed Description: This is generated when you cancel the scan.

Actions: For your information only.

242 Out of memory (size: <1>)

Detailed Description: SanityCheck has run out of memory.

Actions: On the Macintosh, allocate more memory to SanityCheck's partition. On the PC, quit other applications, make sure you have plenty of extra disk space on the disk that hold the virtual memory swap file (boot disk).

243 Error Freeing Memory

Detailed Description: The low level memory allocation routines cannot free memory properly.

Actions: This is most likely a problem with SanityCheck and you should report it Committed Software.

244 Cannot open file '<1>'

Detailed Description: The named disk file cannot be opened.

Actions: Verify that the file should exist, check other errors that are generated to see what might be going on. On windows, verify that you have enough file descriptors, or close other applications to free up file descriptors.

245 Cannot close file.

Detailed Description: The file cannot be closed.

Actions: This is most likely a problem with SanityCheck and you should report it Committed Software.

246 File Position Error (<1>,<2>)

Detailed Description: SanityCheck cannot move to the given position in the file.

Actions: Determine which file is not being positioned properly as it may be damaged.

247 Cannot determine file position.

Detailed Description: An error occurred while trying to find the current position of a file pointer (low level file manipulation code).

Actions: Determine which file is not being positioned properly as it may be damaged.

248 File Read Error (tried: <1>, got: <2>)

Detailed Description: Low level read error.

Actions: See items that occur after this item to determine what actions to take.

249 Unexpected End of file (tried: <1>, got: <2>)

Detailed Description: While reading data, SanityCheck encountered an unexpected end of file.

Actions: See items that occur after this item to determine what actions to take.

250 File Write Error (tried: <1>, wrote: <2>)

Detailed Description: Low level write error.

Actions: See items that occur after this item to determine what actions to take.

251 4D v6 User Constant Type is Duplicated (name: <1>, parent: <2>)

Detailed Description: The given 4DK# resource type has a duplicated name.

Actions: Locate the duplication, and remove it.

252 No equivalent STR# resource for given menu name/item (Name: <1>)

Detailed Description: Your structure file refers to a STR# resource as part of a menu name, but that STR# resource doesn't exist.

Actions: Open the menu and re-type the name, verifying that the STR# resource does indeed exist..

253 Please set the 4D Application for this structure file. You can do this in the Settings:Misc area.

Detailed Description: SanityCheck now requires that you define which 4D Program/Application (4D.exe) that you will use to run this structure file. SanityCheck needs the exact version of 4D you intend to use as it will use various information

Actions: Go to the Settings area, and select the "Misc" tab. Click on the "Set" button in the "4D Application for this structure" area. Go find the correct version of 4D for this structure.

254 No equivalent STR# resource for given Balloon Help (Name: <1>)

Detailed Description: The balloon help specifies that the data is in a STR# resource, however that STR# resource cannot be found by SanityCheck.

Actions: Edit the ballon help and put the correct STR# resource, or add the help text directly.

255 Form/Layout has a malformed header.

Detailed Description: The header of the form/layout is not recognized by SanityCheck and most likely indicates damage to the layout.

Actions: Try deleting the form/layout and re-creating it. If you cannot delete the layout in 4D, you can delete the layout using SanityCheck's Browser.

256 Expected a FORM token but got <1> instead.

Detailed Description: This version 6 form does not begin with a FORM token and is therefore invalid.

Actions: If you also recieved an item [261], please contact Committed Software. Otherwise, delete and recreate the form.

257 Form/Layout is too big. Total size (<1>) should be less than <2>.

Detailed Description: The stored size of the form/layout is larger than the object as stored on disk... something is seriously wrong with this form/layout.

Actions: Delete and recreate the form.

258 Expected a LAYO token but got <1> instead.

Detailed Description: This version 6 form is not in proper version 6 format, as the expected LAYO token is missing.

Actions: If you also recieved an item [261], please contact Committed Software. Otherwise, delete and recreate the form.

259 Expected a version token but got <1> instead.

Detailed Description: This version 6 form is not in proper version 6 format, as the expected version token is missing.

Actions: If you also recieved an item [261], please contact Committed Software. Otherwise, delete and recreate the form.

260 Form/Layout version size should be <1> bytes long but claims to be <2>.

Detailed Description: The size of the version field is invalid for this version 6 form.

Actions: If you also recieved an item [261], please contact Committed Software. Otherwise, delete and recreate the form.

261 Form version is <1> but should be less than <2>

Detailed Description: The form version number is invalid. SanityCheck will attempt to parse the form anyway as oftentimes there is backward compatibility with form versions. However, if you start to get this item number as well as other items that indicate damage to the form, and you have just upgraded to a new version of 4D, then you may need a newer version of SanityCheck.

Actions: Check with Committed Software to determine if you need a new version of SanityCheck.

262 Expected next page to be #<1> but read page #<2>

Detailed Description: SanityCheck expects the pages to be in order (1, 2, 3, etc). If it finds a page out of order, it alerts you to this fact. Future versions of 4D may begin to store the pages out of order, in which case SanityCheck would need to be changed.

Actions: If you also recieved an item [261], please contact Committed Software. Otherwise, delete and recreate the form.

263 Variable '<1>' has a gremlin ('<2>' = <3>).

Detailed Description: The form/layout variable contains a gremlin. A gremlin is a character that is consider to be out of the range of normal characters that are expected to be in ascii text. You can modify what SanityCheck considers a gremlin in the Settings. Gremlins are intended to catch either a typo, or damage to the layout.

Actions: If this is a gremlin, simply open the object in 4D and rename it. If this is a normal character in your language, then change the gremlin map in the Settings for this structure.

264 Variable '<1>' contains a reserved character ('<2>' = <3>).

Detailed Description: These are the current characters considered reserved by SanityCheck: $, Þ, >, <, +, -, /, *, %, &, |, and #. You should not have these characters in the name of a variable.

Actions: Open the form/layout that has this variable and retype the variable name.

265 Database Method List has a malformed header.

Detailed Description: The database method list contains information about the

special methods that are executed by 4D for special events. OnStartup is an example of a Databse Method. The list of database methods has a damaged header and indicates major damage to this structure file.

Actions: Recover from backup.

266 Due to an error condition, Comparison cannot continue.

Detailed Description: Comparison can only occur on files that do not have fatal errors.

Actions: Attend to the noted fatal error and then re-run comparison.

267 Form/layout object is an Plugin/External object, but the name for the Plugin/External is empty.

Detailed Description: The form/layout claims to have a plugin/external object, but the name of the plugin or external is blank.

Actions: Open the layout and edit the name of the plugin/external to make it valid.

268 Form/layout object <1> refers to Plugin/External '<2>' which doesn't appear to exist.

Detailed Description: The named plugin or external doesn't appear to exist.

Actions: Verify that all your plugins/externals are in the proper places (Mac4DX/Win4DX/proc.ext/etc). If they are not present, then put them in their proper places. If they are in their proper places, verify that external is available for the platform you are running on. In the current release of SanityCheck, only the Mac4DX folder is scanned on the Macintosh, and the Win4DX scanned under Windows. You may, therefore, recieve this item if you use an external only on the Macintosh but not under windows. If so, simply ignore this item.

269 NOTE: SanityCheck can attempt to repair this. You can access repair features through Settings:Repair.

Detailed Description: SanityCheck is alterting you to the fact that it can attempt to repair this problem, but that you do not have the repair features activated.

Actions: You can activate the repair features through the Settings area of SanityCheck. Be sure to make a backup of your structure before running any tool that modifies the structure file.

270 There is no list of Form Templates in this structure.

Detailed Description: This version 6 database does not have any Form Templates. This is simply to let you know that the templates do not exist, and all 4D databases seen by SanityCheck developers have had form templates in them. Nothing is necessarily amiss, just different. If you are having bizarre behaviour, these types of warnings might help you pinpoint what is "different" and may be causing the odd behaviour.

Actions: You do not need to do anything.

271 There are <1> lists of Form Templates - there should only be 1.

Detailed Description: Normally, there is one and only one Form template. The extra one is ignored by 4D, but SanityCheck notes it in case it somehow causes problems. It's nothing that should be wrong, but you may want to know that it is different. The form template list is named "FGS#" in the Browser.

Actions: You do not need to do anything.

272 The count of Form templates is too large (is: <1>, should be: <2>).

Detailed Description: SanityCheck computes the maximum size that this form template can be given the amount of space on disk it occupies. It has determined that the claimed size is greater than the computed size, and therefore is invalid. The form template list is named "FGS#" in the Browser.

Actions: Make a backup of your structure. Let SanityCheck fix this item for you.

273 Unexpected end of Form template list.

Detailed Description: Encountered the end of the object before finished parsing the template list. This indicates potential damage to the form template list but might also be related to item [272].

Actions: Make a backup of your structure. Then open the Browser and go to the "FGS#" list. Double click on the proper object in the right list (there should be only one). You should get an error when you open the Browser that complains about running out of data. Try reducing the "Num Lists" number by one, then save, and re-open the item on the right list. Continue this process until you can open the item on the right list without the error appear. Re-run SanityCheck to verify the problem is solved.

274 Form template #<1> has a bad name length of <2>. Should be in the range 0-31.

Detailed Description: The name of the form template is invalid.

Actions: The fastest way to fix this is to open the Browser, select the FGS# list, double click on the object in the right list, and change the name to be valid. Alternatively, you could change it in 4D.

275 Form template named '<1>' (<2>) has no template data.

Detailed Description: The named template has no associated template data (no FGPS). This may cause problems with creating new layouts using the layout wizard.

Actions: Delete the form template in 4D.

276 Form template header is damaged (<1>).

Detailed Description: The form template has a bad header.

Actions: Delete the form template in 4D.

277 There is no list of Pictures in this structure.

Detailed Description: All version 6 structures seen by SanityCheck developers have a list of pictures (even if none exist). This item is here to warn you that something is different in your structure file, although it shouldn't cause any problems at all.

Actions: You do not need to do anything.

278 There are <2> lists of Pictures - there should only be 1.

Detailed Description: There should only be 1 list of pictures in your structure file. The extra lists should be ignored by 4D. This item is here to warn you that something is different in your structure file, although it shouldn't cause any problems at all

Actions: You do not need to do anything.

279 The count of Pictures is too large (is: <1>, should be: <2>).

Detailed Description: SanityCheck computes the maximum size that this picture list can be given the amount of space on disk it occupies. It has determined that the claimed size is greater than the computed size, and therefore is invalid. The form template list is named "PIC#" in the Browser.

Actions: Make a backup of your structure and let SanityCheck fix this for you.

280 Unexpected end of Picture list.

Detailed Description: Encountered the end of the object before finished parsing the template list. This indicates potential damage to the form template list but might also be related to item [279].

Actions: Make a backup of your structure. Then open the Browser and go to the "PIC#" list. Double click on the proper object in the right list (there should be only one). You should get an error when you open the Browser that complains about running out of data. Try reducing the "Num PICTs" number by one, then save, and re-open the item on the right list. Continue this process until you can open the item on the right list without the error appear. Re-run SanityCheck to verify the problem is solved.

281 Picture #<1> has a bad name length of <2>. Should be in the range 0-31.

Detailed Description: The name of the picture is invalid.

Actions: The fastest way to fix this is to open the Browser, select the PIC# list, double click on the object in the right list, and change the name to be valid. Alternatively, you could change it in 4D.

282 Picture named '<1>' (<2>) has no data (the picture doesn't exist in the file).

Detailed Description: The named picture has no associated picture data (no PICT).

Actions: Delete the picture in 4D.

283 Picture list header is damaged (<1>).

Detailed Description: The picture list is damaged.

Actions: Try to open the picture list in 4D and immediately save. If this doesn't work, recover from backup.

284 Header for Picture named '<1>' (<2>) is damaged.

Detailed Description: The picture header is damaged

Actions: Delete the picture in 4D.

285 Picture named '<1>' (<2>) is internally damaged.

Detailed Description: The given picture is damaged internally. Damage can actually be inside the data format of the picture itself. There is no other known tool that detects damage in the PICT data itself (only SanityCheck). Unfortunately, PICT damage can cause your application to crash.

Actions: Re-paste the picture from it's original source. If that did not fix the problem, the original picture may be damaged. In the worse case you can always use a screen capture utility to re-create the image from scratch using a picture of the source.

286 There is no list of Lists in this structure.

Detailed Description: All version 6 structures seen by SanityCheck developers have a list of Lists (even if none exist). This item is here to warn you that something is different in your structure file, although it shouldn't cause any problems at all.

Actions: You do not need to do anything.

287 There are <1> lists of Lists - there should only be 1.

Detailed Description: There should only be 1 list of Lists in your structure file. The extra lists should be ignored by 4D. This item is here to warn you that something is different in your structure file, although it shouldn't cause any problems at all

Actions: You do not need to do anything.

288 The count of Lists is too large (is: <1>, should be: <2>).

Detailed Description: SanityCheck computes the maximum size that this Lists list can be given the amount of space on disk it occupies. It has determined that the claimed size is greater than the computed size, and therefore is invalid. The Lists list is named "LE4D" in the Browser. Note that Lists can either be "EN4D" objects (pre-version 6) or "List" objects (version 6 or greater).

Actions: Make a backup of your structure and let SanityCheck fix this for you.

289 Unexpected end of Lists list.

Detailed Description: Encountered the end of the object before finished parsing the Lists list. This indicates potential damage to the Lists list but might also be related to item [288].

Actions: Make a backup of your structure. Then open the Browser and go to the "LE4D" list. Double click on the proper object in the right list (there should be only one). You should get an error when you open the Browser that complains about running out of data. Try reducing the "Num Lists" number by one, then save, and re-open the item on the right list. Continue this process until you can open the item on the right list without the error appear. Re-run SanityCheck to verify the problem is solved.

290 List #<1> has a bad name length of <2>. Should be in the range 0-31.

Detailed Description: The name of the list is invalid.

Actions: The fastest way to fix this is to open the Browser, select the LE4D list, double click on the object in the right list, and change the name to be valid. Alternatively, you could change it in 4D.

291 List named '<1>' (<2>) has no data (the List doesn't exist in the file).

Detailed Description: The named list has no associated list. Note that the list can be either "EN4D" or "List" in the Browser.

Actions: Verify that the list doesn't exist and delete it if it doesn't.

292 List list header is damaged (<1>).

Detailed Description: The header for the LIsts list is damaged.

Actions: Try to open the List editor in 4D and immediately save. If this doesn't work, recover from backup.

293 Header for List named '<1>' (<2>) is damaged.

Detailed Description: The named List's header is damaged.

Actions: Delete the list in 4D and recreate it.

294 Table/Layout '<1>' has a Trigger - and the Trigger is missing in the structure file (ie, missing <2>).

Detailed Description: The given trigger cannot be found.

Actions: Make a backup of the structure file and allow SanityCheck to fix this problem.

295 Table/File '<1>' field '<2>' points to a non-existant choice list.

Detailed Description: The choice list for the given table/file's field is invalid.

Actions: Open the field with 4D and verify that the choice list is vaild. Alternatively, you can view the data directly in the Browser. Click on the <view> hyperlink and the template for the proper FI4D object will open. Locate the field that has the invalid choice list and change the value to be "0x0000". This will clear the choice list. You can either leave it cleared, or go back into 4D and reselect it. Note if you are really into the templates, you could go find the right EN4D or List ID and enter it directly in the Choice List field. Of course, make backups before using any tool that modifies your structure file.

296 Database Method '<1>' refers to non existant method (<2>).

Detailed Description: The named database method points to a nonexistant method.

Actions: Make a bakcup of your structure and allow SanityCheck to fix this.

297 Internal Error: No free blocks

Detailed Description: This is a serious internal error in SanityCheck.

Actions: On the Macintosh, try increasing the memory partition for SanityCheck. Otherwise, contact Committed Software for more information.

298 Cannot read block (offset: <1>, length: <2>)

Detailed Description: The given block cannot be read from disk. This is a serious error.

Actions: Look to see what other errors are generated as they may be the cause for this low level error. If you cannot fix the problem by attending to related items, recover from backup.

299 Internal Error: Block not found to lock (<1>)

Detailed Description: This is a serious internal error in SanityCheck.

Actions: Contact Committed Software.

300 Internal Error: Block not found to unlock (<1>)

Detailed Description: This is a serious internal error in SanityCheck.

Actions: Contact Committed Software.

301 Bad bitmap block number (got: <1>, max: <2>)

Detailed Description: The bitmap (also called freemap) is how 4D tracks used space on disk. SanityCheck is having trouble locating part of this freemap, and it

may indicate serious damage.

Actions: Look to see what other errors are generated as they may be the cause for this low level error. If you cannot fix the problem by attending to related items, recover from backup.

302 Can't load bitmap block (file position: <1>)

Detailed Description: The bitmap (also called freemap) is how 4D tracks used space on disk. SanityCheck is having trouble locating part of this freemap, and it may indicate serious damage.

Actions: Look to see what other errors are generated as they may be the cause for this low level error. If you cannot fix the problem by attending to related items, recover from backup.

303 Couldn't extend bitmap table

Detailed Description: SanityCheck is attempting to extend the size of the bitmap table (and hence the size of the file on disk). It cannot do so, which may mean the disk/volume is locked or there isn't enough room on the disk to extend the table. NOTE: To compute Bitmap errors, SanityCheck creates a duplicate BITMAP table in the same directory/folder as the structure is located. This file is written to, and needs to be on a writable disk.

Actions: If the structure is on a non-writable disk/volume, then move it to a disk that is writable. Ditto if there's no room left on the disk where the structure exists.

304 Internal: bitmap table cannot handle objects this large (size: <1>, max: <2>)

Detailed Description: There is a limit to the size of an object SanityCheck's internal bitmap table algorithms can handle.

Actions: Contact Committed Software.

305 Cannot find space in file to allocate object (size: <1>)

Detailed Description: SanityCheck is attempting to find space for the given object. It has failed in its attempt and is letting you know about it. This will most likely be in conjunction with a [303] item.

Actions: If you get a [303] item as well, move the structure to a writable disk with extra disk space..

306 Bitmap node in different file segment (offset: <1>, expected seg: <2>)

Detailed Description: SanityCheck should never generate this error, as segments are only for 4D datafiles. However, since the Bitmap code is used between both products, this item still might occur.

Actions: If you see this item, there is serious damage to your structure file and you should recover from backup.

307 Space occupied (offset: <1>, size: <2>)

Detailed Description: The named space is already occupied by another object. This is a low level error generated by the bitmap routines when space is trying to be allocated in a Bitmap. It will most likely be generated by SanityCheck during a scan of a structure file where two objects occupy the same space.

Actions: Look for items that accompany this item and concentrate on fixing those problems first. If this item persists, you should perform 4D Tools:Compact to fix the bitmap table.

308 Cannot extend temporary storage file

Detailed Description: SanityCheck creates a temporary file to create a new bit-map table for comparison against the real bitmap table. This file is stored in the same directory/folder as the structre.

Actions: Move the structure to a disk that has plenty of space and is writable.

309 Object map space is already occupied by another object.

Detailed Description: Part of the object map is claimed by another object. This is serious damage.

Actions: You should perform 4D Tools:Compact.

310 Bitmap overlaps another object on disk (perhaps another bitmap?).

Detailed Description: Part of the bitmap is claimed by another object. This is serious damage.

Actions: You should perform 4D Tools:Compact.

311 Dead space found (pos: <1>, size: <2> bytes)

Detailed Description: Dead space is space that is not used at all by 4D, but is still "allocated" by the bitmap (freemap). 4D tracks space in your structure file using a large bitmap. Sometimes, when an object is deleted, the bitmap is not updated. This means that 4D thinks something exists at this place in the file, but in reality, nothing does. NOTE: when you delete objects with SanityCheck, these items will occur, as SanityCheck does not update the bitmap.

Actions: You do not need to do anything. This does not indicate damage. However, if you get a lot of these and want to reclaim the empty space, simply run 4D Tools:Compact.

312 Space should be claimed (pos: <1>, size: <2> bytes)

Detailed Description: This is a problem waiting to happen: An object claims to be at a particular place on disk, but the bitmap (freemap) claims that that space is unoccupied. This means that in the future, when 4D goes to store another object, it may also use this place, and you will get overlapped objects, which is serious damage.

Actions: Immediately run 4D Tools:Compact.

313 Too many bitmap errors (max: <1>)

Detailed Description: SanityCheck only reports so many bitmap errors (as there could be hundreds of thousands of them). It will report a summary.

Actions: This is for your information.

314 Total dead space found: <1> bytes

Detailed Description: This is the sum of all the [311] items.

Actions: See item [311].

315 Total space not properly claimed by bitmap: <1> bytes

Detailed Description: This is the sum of all the [312] items.

Actions: See item [312].

316 Cannot find the SubTable for field <1><2>

Detailed Description: The field claims to be a SubTable/SubFile field, but the related SubTable/SubFile cannot be found.

Actions: Make a backup and let SanityCheck fix this item.

317 SubTable/SubFile with first field named '<1>' has no parent Table/File. BUT: Also found '<2>' has a bad SubTable/SubFile.

Detailed Description: The found SubTable/SubFile has no parent. But, SanityCheck has also found a field that does't have a SubTable/SubFile. SanityCheck will try to link these two together

Actions: Make a backup of your structure file and let SanityCheck fix this. If you have only one missing SubTable, the link should happen painlessly. However, if you have several missing SubTable pairs, then you need to do some work to determine which are the right pairings. Make copious backups, and work slowly. SanityCheck can tell you the first field of the subtable, and the field of the parent it found, which should help you figure out what to do.

318 The object map is smaller than expected (actual size: <1> stored size: <2>.

Detailed Description: The object map as computed by SanityCheck is different from the one stored on disk.

Actions: Launch and quit 4D. If this does't fix the problem, run 4D Tools:Compact.

319 Orphan object <1>.

Detailed Description: The given object is an "Orphan". Many objects in your structure file require a "parent". For example, for a PICT object to be useful, it must be noted in the PIC# object. If it's not, then there's no way to access the PICT from within 4D - it has no parent - it is an orphan.

Actions: These orphans cause no problems at all for 4D. They just take up disk space, and you can just ignore them if you wish. However, if you want to get rid of them, backup your database. Then run SanityCheck's Browser, locate them, and delete them. Run 4D Tools:Compact to clean up the bitmap and they're gone forever.

320 Table/File <1> has no Forms/Layouts.

Detailed Description: There is no list of forms/layouts for the given Table/File.

Actions: This is perfectly fine to have. Many web-based applications have this scenario. This is for your information only.

321 <1> performed with non-indexed field '<2>' (line <3>).

Detailed Description: A QUERY/SEARCH or other operation is performed with a non-indexed field. This item is generated to help you pinpoint areas that might be speeded up with the addition of an index. However, SanityCheck cannot tell you where is most optimal for your database to place an index, you must decide. SanityCheck is only noting that the field you have used is not indexed, and you may consider putting and index on that field. You could also try to index the field, run this section of code and see if it signifcantly speeds up performance. If it does, then run your application in general, doing normal operations. If you see no slowdown due to the added index, then you may as well keep it.

Actions: You do not need to do anything. You may also decide to index this field.

322 LIBA object is missing.

Detailed Description: The LIBA object is the Menu Bar List. Every structure file should have one. If yours doesn't, then something is seriously wrong.

Actions: Try to open the Menu editor in 4D. If you can open it, close and quit 4D. If that doesn't fix it, then recover from backup.

323 Syntax: Expected '<1>' after conjunction operator on line <2>.

Detailed Description: This is warning that a build search or query is not in full conformance with the stated 4D syntax. For example, "QUERY([Table1]; & [Table1]Field1=1)" is not correct but "QUERY([Table1]; & ;[Table1]Field1=1)" is correct. The difference is the semicolon after the "&" (conjunction operator). Some developers have noted that leaving the semicolon absent will cause erroneous results from QUERY/SEARCH. Committed Software has not confirmed this behaviour, but implemented this check at the request of developers who noted the issue

Actions: Might as well be safe and add the semi-colon.

324 Divide by constant zero on line <1>.

Detailed Description: SanityCheck has found a division by zero in your code. Probably shouldn't try to do that.

Actions: Edit the line and remove the divide by zero.

325 Variable is a reserved word ('<1>').

Detailed Description: The variable on this Form/Layout has a name that is reserved. This is to catch situations where you might name a variable "ABORT" which is also a 4D command.

Actions: Change the name of the variable.

326 Field '<1>': choice list active, but no choice list is chosen

Detailed Description: The choice list checkbox is selected, but no choice list is chosen in the choice list. This shouldn't cause any problems, but you should uncheck the checkbox.

Actions: Uncheck the checkbox.

327 Style Sheet has a malformed header

Detailed Description: The style sheet cannot be read due to a bad style sheet header.

Actions: Try to delete the stylesheet, then recreate it. If 4D does not allow you to do this, use the SanityCheck “browser” to delete the style sheet.

328 Unexpected end of object encountered in Style Sheets

Detailed Description: The style sheet is damaged (and unrecoverable)

Actions: Delete the style sheet and re-create it.

329 Style Sheet has no name

Detailed Description: The style sheet has no name.

Actions: Edit the style sheet in 4D and enter a name.

330 Style Sheet name is too long (size: <1>, max: <2>)

Detailed Description: The name of the style sheet is too long. This may indicate general damage in the style sheet.

Actions: Re-edit the name and shorten the name. If that does not work, delete and recreate the style sheet.

331 Style Sheet name has gremlin ('<1>' = <2>)

Detailed Description: The name of the style sheet has an unprintable character (gremlin).

Actions: Edit the name of the style sheet and modify the name. If the name is indeed OK (perhaps you are using another langauge), make sure your gremlin map is set up properly (File: Settings: Gremlin).

332 Style Sheet Mac Font Name is too long (size: <1>, max: <2>).

Detailed Description: The Macintosh font name in the Style sheet is too long.

Actions: Edit the Mac Font name and shorten it.

333 Style Sheet Mac Font Name has gremlin ('<1>' = <2>)

Detailed Description: The Macintosh font name has a nonprintable (gremlin) character in it.

Actions: Edit the Mac font name and remove the gremlin, or modify your gremlin settings to accomodate your language (File Menu: Settings: Gremlin).

334 Style Sheet Win Font Name is too long (size: <1>, max: <2>)

Detailed Description: The Windows Font Name in the Style sheet is too long.

Actions: Edit the Windows font name and shorten it.

335 Style Sheet Win Font Name has gremlin ('<1>' = <2>)

Detailed Description: The Windows font name has a nonprintable (gremlin) character in it.

Actions: Edit the Windows font name and remove the gremlin, or modify your gremlin settings to accomodate your language (File Menu: Settings: Gremlin).

336 Style Sheet Win95 Font Name is too long (size: <1>, max: <2>)

Detailed Description: The Windows95/98 Font Name in the Style sheet is too long.

Actions: Edit the Windows font name and shorten it.

337 Style Sheet Win95 Font Name has gremlin ('<1>' = <2>)"

Detailed Description: The Windows95/98 font name has a nonprintable (grem-

lin) character in it.

Actions: Edit the Windows 95/98 font name and remove the gremlin, or modify your gremlin settings to accomodate your language (File Menu: Settings: Gremlin).

338 Variable '<1>' on line <2> has the same name as a Konstant (4DK# constant).

Detailed Description: The named variable has the same name as a constant and it is therefore unclear which is correct to use.

Actions: Change the name of the variable.

339 4D v6 User Constant Type is too long (name: <1>, parent: <2>)

Detailed Description: The constant name is too long.

Actions: Edit the 4DK# resource and change the name.

340 Method name '<1>' contains a gremlin ('<2>' = <3>)

Detailed Description: The method name contains a non-printable character (gremlin).

Actions: Edit the name of the method and re-type it to remove the non-printable character or modify your gremlin settings to accomodate your language (File Menu: Settings: Gremlin)

341 Style Sheet '<1>' contains the same ID as another style sheet (ID=<2>)

Detailed Description: The two StyleSheets have the same ID. This will cause problems when determining which Style Sheet to use.

Actions: Set the ID for the second Style Sheet. You can use the SanityCheck “Browser” to find the style sheets and edit them.

342 CC4D object <1> used by two or more methods (<1>, <2>)

Detailed Description: The same CC4D object (resource) is being claimed by two methods. This should never happen.

Actions: Delete all methods that claim this object. Recreate the methods.

343 Form List (TF4D) for table <1> has no Output Form defined

Detailed Description: The given list of formst has no output form defined. 4D Tools will sometimes tag this situation as “damaged” even though there is no damage.

Actions: Add a default Output Form.

344 Form List (TF4D) for table <1> has invalid Output Form (value: <2>, max: <3>)

Detailed Description: The Output Form listed in the form list is invalid.

Actions: Select a proper Output Form for this table.

345 Form List (TF4D) for table <1> has no Input Form defined

Detailed Description: The input form is not defined.

Actions: Using 4D, open up the Explorer, select the “Form” tab, select the given Table, select the form you wish to have as the default “Input” form, and then select the “Input Form” checkbox. An “I” (eye) will appear to the right of the form, indicating it is the default input form.

346 Form List (TF4D) for table <1> has invalid Input Form (value: <2>, max: <3>)

Detailed Description: The given input form is invalid.

Actions: Using 4D, open up the Explorer, select the “Form” tab, select the given Table, select the form you wish to have as the default “Input” form, and then select the “Input Form” checkbox. An “I” (eye) will appear to the right of the form, indicating it is the default input form.

347 '<1>' never explicitly referenced/called by another Method, Form, or menu item

Detailed Description: The given object is never explicitly referenced by another method or form or menu item. This is not necessarily a problem. For example, if you use this code: EXECUTE(“myMethod”); then SanityCheck will not recognize that “myMethod” has been called.

Actions: No action is necessary. You may, however, choose to delete those objects that you are sure are not being used by your project. Note that even though this item appears, the object may actually still be used by the structure.

348 '<1>' called by '<2>' with more parameters than defined (called: <3>, defined: <4>).

Detailed Description: The called method (<1>) is being called with more parameters than have been explicitly defined (using compiler directives such as C_INTEGER). This item is only reported if “count parameters” is never used in the called method.

Actions: This is not necessarily a problem, but if you are consistant about defining all parameters to your methods, this may indicate a problem. You should look at the calling method (<2>) and make sure that it is calling the method properly. You should check the called method (<1>) and verify the parameters are defined properly.

349 '<1>' called by '<2>' with less parameters than defined (called: <3>, defined: <4>)

Detailed Description: The called method (<1>) is being called with less parameters than have been explicitly defined (using compiler directives such as C_INTEGER). This item is only reported if “count parameters” is never used in the called method.

Actions: This is not necessarily a problem, but if you are consistant about defining all parameters to your methods, this may indicate a problem. You should look at the calling method (<2>) and make sure that it is calling the method properly. You should check the called method (<1>) and verify the parameters are defined

properly.

350 '<1>' called by '<2>' with more parameters than used in the <M> (called: <3>, used: <4>)

Detailed Description: The called method (<1>) is being called with more parameters than are being used int he method. This item is only reported if “count parameters” is never used in the called method.

Actions: This is not necessarily a problem, but may indicate a programming error. You should look at the calling method (<2>) and make sure that it is calling the method properly. You should check the called method (<1>) and verify it is using it’s parameters properly.

351 '<1>' called by '<2>' with less parameters than used in the method (called: <3>, used: <4>)

Detailed Description: The called method (<1>) is being called with less parameters than are being used int he method. This item is only reported if “count parameters” is never used in the called method.

Actions: This is probably a programming error. In the least, it is dangerous as you may try to access a parameter that hasn’t been passed. You should look at both methods (the caller and called) and verify that this is coded properly.

352 '<1>' uses more parameters than actually defined in the method(defined: <2>, used: <3>)

Detailed Description: The given method is using (referencing) more parameters than are defined. For example, if your code defines $1 and $2 (C_INTEGER($1;$2), but then makes references to $3, this item should be generated.

Actions: If you are consistant about defining your parameters, then define the missing parameters. Otherwse, disable this item in preferences.

353 '<1>' uses less parameters than actually defined in the <M> (defined: <2>, used: <3>)

Detailed Description: The given method is using (referencing) fewer parameters than are defined. For example, if your code defines $1 and $2 (C_INTEGER($1;$2), but then you never reference $2, this item should be generated.

Actions: If you are consistant about defining your parameters, then remove definitions of the unused parameters (and change your internal documentation!). Otherwse, disable this item in preferences.

354 '<1>' uses more parameters than actually defined in the COMPILER method (defined: <2>, used: <3>)

Detailed Description: If you use a COMPILER_ method, you may define the parameters that a method uses. This item is generated if there is a descrepency between what is defined in the compiler method and what is used by the method.

Actions: Most likely, you'll want to add more parameter definitions to your COMPILER_ method. If you are not consistant about your COMPILER_ definitions for parameters, then simply disable this item in preferences.

355 '<1>' uses less parameters than actually defined in the COMPILER method (defined: <2>, used: <3>)

Detailed Description: If you use a COMPILER_ method, you may define the parameters that a method uses. This item is generated if there is a descrepency between what is defined in the compiler method and what is used by the method.

Actions: If you wish to be consistant regarding the parameters defined in the COMPILER_ method and those in the actual method, then either remove the extra definition from the COMPILER_ method, or add a reference to the extra parameter in the method..

356 Cannot read header of Components definition

Detailed Description: The component header cannot be read (it appears to be damaged).

Actions: De-install and re-install the component.

357 Component definition object is too small (size: <1>, minimum:<2>)

Detailed Description: The component is most likely damaged.

Actions: De-install and re-install the component.

358 A text object within the Component definition is malformed. Cannot read the component.

Detailed Description: Some text within the component is malformed. The Component cannot be read.

Actions: De-install and re-install the component.

359 The list for type <1> is malformed. Cannot read the component.

Detailed Description: The list for the given type of object within the component is damaged.

Actions: De-install and re-install the component.

360 The component claims to own object <1>, but it does not exist in the structure.

Detailed Description: The component is making a reference to an object that shoud be in the structure, but the object does not appear.

Actions: De-install and re-install the component. If this does not fix the problem, contact the component author.

361 Object <1> has an unknown state (public/protected/private) (valid: <2>, state: <3>)

Detailed Description: Component objects can have one of three states: public/protected/private. This object has an unknown state and is probably damaged.

Actions: De-install and re-install the component. If this does not fix the problem, contact the component author.

362 Object <1> is encrypted as if it were part of a component

Detailed Description: The given object appears to be encrypted (compressed) as if it were part of a component. However, no component claims to own this object.

Actions: De-install and re-install ALL components. Try to determine where the object may have come from, and contact the author of that component or program.

363 '<1>': No Style Sheet is used by this object

Detailed Description: This is not a problem with your structure. SanityCheck will warn you of all objects that do not use style sheets. This is very useful if you are building a cross platfrom database.

Actions: If you want to make sure that every object has a style sheet, then find the object and attach a style sheet. If you do not care about style sheets, disable this item in preferences.

364 '<1>': Style Sheet ID #<2> cannot be found in list of StyleSheets

Detailed Description: The given style sheet was referenced but it cannot be found.

Actions: Within 4D, re-assign the style sheet for this object..

365 '<1>': Variable defined in compiler statement but never referenced in code

Detailed Description: The given variable (process/interprocess) was defined in a compiler statement but never referenced in code (in a method or form).

Actions: You may actually be using this variable in other ways (for example, via an EXECUTE call. If you are sure you are not using it, you might as well delete the compiler reference to it.

366 Variable '<1>' in form has a 31 characters. Tech Tip suggests only 30 chars for layout variables

Detailed Description: A variable in a form should only have 30 chars for its name. There is a TechTip that shows how having 31 chars could cause problems within the form.

Actions: Change the name of the variable to be only 30 characters.

367 List of Forms for Table '<1>' appears to be compressed, and should not be

Detailed Description: The list of forms should never be compressed, yet it appears to be.

Actions: Download the most recent version of 4D Tools and run Compact. This was a result of an earlier bug in 4D Insider which has since been fixed.

328 This structure has an internal version of v3 of 4D, but appears to be a v6 file. Cannot parse Data Dictionary

Detailed Description: The data dictionary is unparsable.

Actions: Within v6 of 4D, open the Database Structure, modify something (remove an index). Quit 4D, restart 4D, Open the Database Structure, re-add the index. Quit 4D. Re-run SanityCheck.

369 Plugin '<1>' cannot handle it's own error code (error code: <2>)."

Detailed Description: The SanityCheck Plugin (not a 4D Plugin) is not responding to reporting it's own error code.

Actions: Contact that SanityCheck plugin author.

370 Variable '<1>' already defined in '<2>' on line <3> with a different type

Detailed Description: The variable was already defined elsewhere with a different type.

Actions: Decide which definition of the variable is accurate and either remove one or make them both the same.

371 Parameter '<1>' of method '<2>' already defined with a different type

Detailed Description: The parameter was already defined elsewhere with a different type. This may be either in the same method or in a COMPILER_ method

Actions: Decide which definition of the parameter is accurate and either remove one or make them both the same.

372 The method '<1>' (used as a string parameter) does not exist (line <2>)

Detailed Description: If you use routines such as "EXECUTE ON SERVER" or "EXECUTE ON CLIENT", you should pass the name of the method as a string to these methods. SanityCheck verifies that these methods actually exist. This item is generated if SanityCheck cannot find the method.

Actions: Make sure the method exists. Note: you could fool SanityCheck if you have something like this: EXECUTE ON SERVER(("mymethod")+"1"). In which case you should just ignore this item or remove the superflous parens around "mymethod".

373 The McCabe complexity of this method is <1>.

Detailed Description: This is not a problem. See the section on McCabe com-

plexity for a discussion on this topic.

Actions: If you choose to do so, you may attempt to reduce the McCabe complexity of this routine. Otherwise, you may choose to increase the Tolerance for the complexity via Settings:Tolerances.

374 Variable '<1>' has an average span of <2>.

Detailed Description: Please see the section on Variable Span / Live Time for detailed discussion of the meaning of average span. This item is informing you that this variables average span is greater than the tolerance set up.

Actions: Either ignore this item, re-arrange the method, or increase the tolerance via Settings:Tolerances.

375 Variable '<1>' has a live time of <2>

Detailed Description: Please see the section on Variable Span / Live Time for detailed discussion of the meaning of average span. This item is informing you that this variables live time is greater than the tolerance set up.

Actions: Either ignore this item, re-arrange the method, or increase the tolerance via Settings:Tolerances.

376 Average of all variable average spans: <1>

Detailed Description: Please see the section on Variable Span / Live Time for detailed discussion of the meaning of average span. This item is informing you of the average of all the average spans. This sounds confusing, but it is the average of item [374] for all variables in this method.

Actions: Either ignore this item, re-arrange the method, or increase the tolerance via Settings:Tolerances.

377 Average of all variable live time: <1>

Detailed Description: Please see the section on Variable Span / Live Time for detailed discussion of the meaning of average span. This item is informing you of the average of all the variable live times for this method. This sounds confusing, but it is the average of item [375] for all variables in this method.

Actions: Either ignore this item, re-arrange the method, or increase the tolerance via Settings:Tolerances.

378 <1> Total McCabe Complexity (sum of all Methods).

Detailed Description: This is the sum of all McCabe complexities of the methods. Given that there are probably a large number of short scripts (for forms object methods), this is probably not very useful.

Actions: Just for your information.

379 <1> Maximum McCabe Complexity

Detailed Description: This is the maximum McCabe complexity found in the project.

Actions: Just for your information.

380 <1> Average (mean) McCabe Complexity.

Detailed Description: This is the average of all McCabe complexities of the methods. Given that there are probably a large number of short scripts (for forms object methods), this is probably not very useful.

Actions: Just for your information.

381 Object for last error [307] is <1>

Detailed Description: This is a follow on item which gives more information when an item [307] is encountered. It helps you determine which object appears to be having problems vis-a-vis item [307].

Actions: Follow directions for item [307] with the extra information this item provides.

382 <1>: Variable referenced in method <2> but never declared in a compiler statement.

Detailed Description: This option helps you use the 4D language as if it were a strongly typed language. It warns you whenever a variable has been referenced but never explicitly defined.

Actions: Either declare the variable or ignore this item.

383 <1>: Variable referenced in form <2> but never declared in a compiler statement.

Detailed Description: The same as item 382, except broken out for those variables that are only referenced in forms. The reason for this is that many developer do not care if variables only used in forms are typed or not. This option with option [382] allow you to more fully force your coding style to be strongly typed.

Actions: Either declare the variable or ignore this item.

384 Variable <1> already declared in <2> on line <3>

Detailed Description: This option warns you of multiple delcaration, even if the declaration is of the same type. Many developers have requested a way to know if a varaible is declared in more than one place (even if it's the same type in all places).

Actions: Delete one of the two declarations, or ignore this item.

385 Function '<1>' uses default table on line <2>

Detailed Description: Many developers want to know when they are relying upon the DEFAULT TABLE call, as they can then track down and clean up any references to DEFAULT TABLE. This option allows you to accomplish this task by informing you when you use a 4D function that appears to rely upon the DEFAULT TABLE. Warning: there are some instances when SanityCheck cannot determine if the function is using a default table or not. Specifically if you are using pointers

to tables. This does make this option less reliable for some coding styles.

Actions: Either add the specific file reference, or ignore this item if you are using DEFAULT TABLE.

386 DEFAULT TABLE is used on line <1>

Detailed Description: As with item [385], this item is attempting to point out places where default tables are used.

Actions: If you are using DEFAULT TABLE in your database and wish to continue doing so, do nothing. If you are trying to remove uses of DEFAULT TABLE, you should first carefully track down all functions that are relying upon the DEFAULT TABLE (see item [385] for help). Once you have done this, you can then remove the DEFAULT TABLE. Do be careful to ensure that no code is reliant upon the DEFAULT TABLE before you remove it, or you may end up with non-functioning code.

387 Inherited form has a bad table number (table number: <1>, form: <2>)

Detailed Description: The form is inheriting from another form, yet the table number is invalid.

Actions: Using 4D, re-assign the inherited form. If this does not work, delete and re-create the sub-form. If this does not work, delete the parent form and recreate both the parent and the child.

388 Form object <1> has a string resource label <2> which doesn't exist.

Detailed Description: The object has a string resource label which does not appear to exist in any of the resources in any related file (structure file, 4D, plugins, etc).

Actions: This may not be a problem if you specifically load a resource file using 4D to resolve these items. If you do this, then you should ignore this item. Otherwise, you should open the form and re-assign the string resource.

389 <1>: Default Style Sheet is used by this offscreen object.

Detailed Description: This item is the same as item [363], but is specifically for offscreen objects. This allows developers who put buttons offscreen to not be bothered with setting style sheets for the offscreen objects. Currently, SanityCheck can only know whether an object is onscreen or offscreen if you use the "Set Size" option for "Size based on" in the "Resizing Options" of the form in question. SanityCheck does cannot yet use the "Based on object xxx" option.

Actions: If you put objects offscreen because you will move them onscreen from your code, then you may wish to assign specific style sheets to this item. Otherwise, you probably just want to ignore this item.

Index

Symbols

Numerics

A

B

C

D

E

F

H

I

L

T

U

V

W

Z